- Miscellaneous
  "The *sewer* overflowed." vs. "a *sewer* is not a tailor."
  "He *moped* since his parents refused to buy a *moped*."
  "*Agape* is a Greek word." vs. "His mouth was *agape*."

As discussed earlier, abbreviation/acronym expansion and linguistic analysis described in Sections 14.4.1 and 14.5 are two main sources of information for TTS systems to resolve homograph ambiguities.

We close this section by introducing two special sources of pronunciation ambiguity that are not fully addressed by current TTS systems. The first one is a variation of dialects (or even personal dialect—*idiolect*). For example, some might say *tom[ey]to*, while some others might say *tom[aa]to*. Another example is that Boston natives tend to reduce the */r/* sound in sentences like "Park your car in Harvard yard." Similarly, some people use the spelling pronunciation *in-ter-es-ting* as opposed to *intristing*. Finally, speech rate and formality level can influence pronunciation. For example, the */g/* sound in *recognize* may be omitted in faster speech. It might be a sensible decision to output all possible pronunciations as a multiple pronunciation list and hope the synthesis back end picks the one with better acoustic/prosodic voice rendition. While true homographs may be resolved by linguistic and discourse analysis, achieving a consistent presentation of dialectal and stylistic variation is an even more difficult research challenge.

The other special source of ambiguity in TTS is somewhat different from what we have considered so far, but may be a concern in some markets. Most borrowed or foreign single words and place names are realized naturally with pronunciation normalized to the main presentation language. Going beyond that, language detection refers to the ability of a TTS system to recognize the intended language of a multiword stretch of text. For example, consider the fragment "Well, as for the next department head, that is simply *une chose entendue*." The French phrase "*une chose entendue*" (something clearly understood) might be realized in a proper French accent and phone pronunciation by a bilingual English/French reader. For a TTS system to mimic the best performance, the system must have:

- language identification capability
- dictionaries and rules for both languages
- voice rendition capability for both languages

## 14.7. MORPHOLOGICAL ANALYSIS

General issues in morphology are covered in Chapter 2. Here, we consider issues of relating a surface orthographic form to its pronunciation by analyzing its component morphemes, which are minimal, meaningful elements of words, such as prefixes, suffixes, and stem words themselves. This decomposition process is referred as *morphological analysis* [28]. When a dictionary does not list a given orthographic form explicitly, it is sometimes possible to analyze the new word in terms of shorter forms already present. These shorter forms

EX-1018
U.S. Patent 10,510,341

may combine as prefixes, one or more stems or roots, and suffixes to generate new forms. If a word can be so analyzed, the listed pronunciations of the pieces can be combined, perhaps with some adjustment (phonological rules), to yield a phonetic form for the word as a whole.

The prefixes and suffixes are generally considered bound, in the sense that they cannot stand alone but must combine with a stem. A stem, however, can stand alone. A word such as *establishment* may be decomposed into a "stem" *establish* and a suffix *-ment*. In practice, it is not always clear where this kind of analysis should stop. That is, should a system attempt to further decompose the stem *establish* into *establ* and *-ish*? These kinds of questions ultimately belong to etymology, the study of word origins, and there is no final answer. However, for practical purposes, having three classes of entries corresponding to prefixes, stems, and suffixes, where the uses of the affixes are intuitively obvious to educated native speakers, is usually sufficient. In practical language engineering, a difference that makes no difference is no difference, and unless there is a substantial gain in compression or analytical power, it is best to be conservative and list only obvious and highly productive affixes.

The English language presents numerous genuine puzzles in morphological analysis. For example, there is the issue of abstraction: is the word *geese* one morpheme, or two (base *goose* + abstract pluralizing morpheme)? For practical TTS systems, relying on large dictionaries, it is generally best to deal with concrete, observable forms where possible. In such a lexically oriented system, the word geese probably should appear in the lexicon as such, with attached grammatical features including plurality. Likewise, it is simpler to include *children* in the lexical listing rather than create a special pluralizing suffix *-ren* whose use is restricted to the single base *child*.

The morphological analyzer must attempt to cover an input word in terms of the affixes and stems listed in the morphological lexicon. The covering(s) proposed must be legal sequences of forms, so that often a word grammar is supplied to express the allowable patterns of combinations. A word grammar might, for example, restrict suffixation to the final or rightmost stem of a compound, thus allowing plurality on the final element of *businessmen* but not in the initial stem (*businessesman*). In support of the word grammar, all stems and affixes in the lexicon would be listed with morphological combinatory class specifications, usually subtyped in accordance with the base POS categories of the lexicon entries. That is, verbs would typically accept a different set of affixes than nouns or adjectives. In addition, spelling changes that sometimes accompany affixation must be recognized and undone during analysis. For example, the word *stopping* has undergone final consonant doubling as part of accommodating the suffix *ing*.

A morphological analysis system might be as simple as a set of *suffix-stripping* rules for English. If a word cannot be found in the lexicon, a suffix-stripping rule can be applied to first strip out the possible suffix, including *–s, -'s, -ing, -ed, -est, -ment*, etc. If the stripped form can be found in the lexicon, a morphological decomposition is attained. Similarly, *prefix-stripping* rules can be applied to find prefix-stem decomposition for prefixes like *in-, un-, non-, pre-, sub-, etc.*, although in general prefix stripping is less reliable.

Suffix and prefix stripping gives an analysis for many common inflected and some derived words such as *helped, cats, establishment, unsafe, predetermine, subword*, etc. It helps in saving system storage. However, it does not account for compounding, issues of legality

of sequence (word grammar), or spelling changes. It can also make mistakes (from a synchronic point of view: *basement* is not *base* + *-ment*), some of which will have consequences in TTS rendition. A more sophisticated version could be constructed by adding elements such as POS type on each suffix/prefix for a rudimentary legality check on combinations. However, a truly robust morphological capability would require more powerful formal machinery and a more thorough analysis. Therefore, adding irregular morphological formation into a system dictionary is always a desirable solution.

Finally, sometimes in commercial product names the compounding structure is signaled by word-medial case differences, e.g., AltaVista™, which can aid phonetic conversion algorithms. These can be treated as two separate words and will often sound more natural if rendered with two separate main stresses. This type of decomposition can be expanded to find compound words that are formed by two separate nouns. Standard morphological analysis algorithms employing suffix/prefix stripping and compound word decomposition are summarized in Algorithm 14.4. Note that the algorithm can be easily modified to handle words constructed by a combination of prefix, suffix, and compound.

**ALGORITHM 14.4: *MORPHOLOGICAL ANALYSIS***

1. **Dictionary Lookup**
   Look up word w in lexicon
   **If** found
      Output attributes of the found lexical entry and exit
2. **Suffix Stripping**
   If word ends in *-s, -'s, -ing, -ed, -est, -ment,* etc.
      Strip the suffix from word w to form u
   **If** stripped form u found in lexicon
      Output attributes of the stem and suffix and exit
3. **Prefix Stripping**
   If word begins with *in-, un-, non-, pre-, sub-,* etc.
      Strip the prefix from word w to form u
   **If** stripped form u found in lexicon
      Output attributes of the prefix and stem and exit
4. **Compound word decomposition**
   **If** detect word-medial case differences within word w
      Break word w into a multiple words $u_1$, $u_2$, $u_3$, ...according to case changes
      **For** words $u_1$, $u_2$, $u_3$, **goto 1.**
   **Else if** word w can be decomposed into two nouns $u_1$, $u_2$ in lexicon
      Output attributes of the $u_1$, $u_2$ and exit
5. Pass word w to letter-to-sound module

## 14.8. LETTER-TO-SOUND CONVERSION

The best resource for generating (symbolic) phonetic forms from words is an extensive word list. The accuracy and efficiency of such a solution is limited only by the time, effort, and knowledge brought to bear on the dictionary construction process. As described in Section 14.2, a general lexicon service is a critical resource for the TTS system. Thus, the first and the most reliable way for grapheme-to-phoneme conversion is via dictionary lookup.

Where direct dictionary lookup fails, rules may be used to generate phonetic forms. Under earlier naïve assumptions about the regularity and coverage of simple descriptions of English orthography, rules have traditionally been viewed as the primary source of phonetic conversion knowledge, since no dictionary covers every input form and the TTS system must always be able to speak any word. A general *letter-to-sound* (LTS) conversion is thus required to provide phonetic pronunciation for any sequence of letters.

Inspired by the phonetic languages, letter-to-sound conversion is usually carried out by a set of rules. These rules can be thought of as dictionaries of fragments with some special conventions about lookup and context. Typically, rules for phonetic conversion have mimicked phonological rewriting in phonological theory [5], including conventions of ordering, such as *most specific first*. In phonological rules, a target is given and the rewrite is indicated, with context following. For example, a set of rules that changes orthographic *k* to a velar plosive */k/* except when the *k* is word-initial ('[') followed by *n* might appear as:

```
k -> /sil/ % [ _ n
k -> /k/
```

The rule above reads that *k* is rewritten as (phonetic) silence when in word initial position and followed by *n*, otherwise *k* is rewritten as (phonetic) */k/*. The underscore in the first line is a placeholder for the *k* itself in specifying the context. This little set properly treats *k* in *knight*, *darkness*, and *kitten*. These are formally powerful, context-sensitive rules. Generally a TTS system require hundreds or even thousands of such rules to cover words not appearing in the system dictionary or *exception list*. Typically rules are specified in terms of single-letter targets, such as the example for *k* above. However, some systems may have rules for longer fragments, such as the special vowel and consonant combinations in words like *neighbor* and *weigh*. In practice, a binary format for compression, a corresponding fragment matching capability, and a rule index must be defined for efficient system deployment.

Rules of this type are tedious to develop manually. As with any *expert system*, it is difficult to anticipate all possible relevant cases and sometimes hard to check for rule interference and redundancy. In any case, the rules must be verified over a test list of words with known transcriptions. Generally, if prediction of main stress location is not attempted, such rules might account for up to 70% of the words in a test corpus of general English. If prediction of main stress is attempted, the percentage of correct phonetic pronunciations is much lower, perhaps below 50%. The correct prediction of stress depends in part on morphology, which is not typically explicitly attempted in this type of simple rule system (though fragments corresponding to affixes are frequently used, such as *tion -> /ah ax n/*). Certainly, such rules can be made to approach dictionary accuracy, as longer and more explicit mor-

phological fragments are included. One extreme case is to create one specific rule (containing exact contexts for the whole word) for each word in the dictionary. Obviously this is not desirable, since it is equivalent to putting the word along with its phonetic pronunciation in the dictionary.

In view of how costly it is to develop LTS rules, particularly for a new language, attempts have been made recently to automate the acquisition of LTS conversion rules. These self-organizing methods believe that, given a set of words with correct phonetic transcriptions (the offline dictionary), an automated learning system could capture significant generalizations. Among them, classification and regression trees (CART) have been demonstrated to give satisfactory performances for letter-to-sound conversion. For basic and theoretic description of CART, please refer to Chapter 4.

In the system described in [14], CART methods and phoneme trigrams were used to construct an accurate conversion procedure. All of the experiments were carried on two databases. The first is the NETALK [25], which has hand-labeled alignment between letter and phoneme transcriptions. The second is the CMU dictionary, which does not have any alignment information. The NETALK database consists of 19,940 entries, of which 14,955 were randomly selected as a training set and the remaining 4951 were reserved for testing. Those 4951 words correspond to 4985 entries in the database because of multiple pronunciations. The hand-labeled alignments were used directly to train the CART for LTS conversion. The CMU dictionary has more than 100,000 words, of which the top 60,000 words were selected based on unigram frequencies trained from North American Business News. Among them, 52,415 were used for training and 9719 reserved for testing. Due to multiple pronunciations, those 9719 words have 10,520 entries in the dictionary. Due to lack of alignment information, dynamic programming was used to align each letter to the corresponding phoneme before training the LTS CART.

The basic CART component includes a set of *yes-no* questions and a procedure to select the best question at each node to grow the tree from the root. The basic *yes-no* question for LTS conversion looks like "*Is the second right letter 'p'?*" or "*Is the first left output phoneme /ay/?*" The questions for letters could be on either the left or the right side. For phones, only questions on the left side were used, for simplicity. The range of question positions must be long enough to cover the long-distance phonological variations. It was found that the 11-letter window (5 for left letter context and 5 for right letter context) and 3-phoneme window for left phoneme context are generally sufficient. A primitive set of questions would be the set of all the singleton questions about each letter or phoneme identity. When growing the tree, the question that had the best entropy reduction was chosen at each node. We observed that if we allow the node to have a complex question that is a combination of primitive questions, the depth of the tree will be greatly reduced and the performance improved. For example, the complex question "*Is the second left letter 't' and the first left letter 'i' and the first right letter 'n'?*" can capture '*o*' in common suffix "*tion*" and convert it to the right phoneme. Complex questions can also alleviate the data fragment problem caused by greedy nature of the CART algorithm. This way of finding such complex questions is similar to those used in Chapter 4. The baseline system built using the above techniques has error rates as listed in Table 14.11.

**Table 14.11** LTS baseline results using CART [13].

| Database | Phoneme | Word |
|---|---|---|
| CMU Lexicon | 9.7% | 35.0% |
| NETTALK | 9.5% | 42.3% |

The CART LTS system [14] further improved the accuracy of the system via the following extensions and refinements:

- *Phoneme trigram rescoring*: A statistical model of phoneme co-occurrence, or phonotactics, was constructed over the training set. A phonemic trigram was generated from the training samples with back-off smoothing, and this was used to rescore the $n$-best list generated by LTS.
- *Multiple tree combination:* The training data was partitioned into two parts and two trees were trained. When the performance of these two trees was tested, it was found that they had a great overlap but also behaved differently, as each had a different focus region. Combining them together greatly improved the coverage. To get a better overall accuracy, the tree trained by all the samples was used together with two other trees, each trained by half of the samples. The leaf distributions of three trees were interpolated together with equal weights and then phonemic trigram was used to rescore the $n$-best output lists.

By incrementally experimenting with addition of these extensions and refinements, the results improved, as shown in Table 14.12.

These experiments did not include prediction of stress location. Stress prediction is a difficult problem, as we pointed out earlier. It requires information beyond the letter string. In principle, one can incorporate more lexical information, including POS and morphologic information, into the CART LTS framework, so it can be more powerful to learn the phonetic correspondence between the letter string and lexical properties.

**Table 14.12** LTS using multiple trees and phonemic trigram rescoring [13].

| Database | Phoneme | Word |
|---|---|---|
| CMU Lexicon | 8.2% | 26.9% |
| NETTALK | 8.1% | 34.2% |

## 14.9. EVALUATION

Ever since the early days of TTS research [21, 31], evaluation has been considered an integral part of the development of TTS systems. End users and application developers are

mostly interested in the end-to-end evaluation of TTS systems. This *monolithic* type of whole-system evaluation is often referred to as *black-box* evaluation. On the other hand, *modular* (component) testing is more appropriate for TTS researchers when working with isolated components of the TTS system, for diagnosis or regression testing. We often refer to this type of evaluation as *glass-box* evaluation. We discuss the modular evaluations in each modular TTS chapter, while leaving the evaluation of the whole system to Chapter 16.

For text and phonetic analysis, automated, analytic, and objective evaluation is usually feasible, because the input and output of such module is relatively well defined. The evaluation focuses mainly on symbolic and linguistic level in contrast to the acoustic level, with which prosodic generation and speech synthesis modules need to deal. Such tests usually involve establishing a test corpus of correctly tagged examples of the tested materials, which can be automatically checked against the output of a text analysis module. It is not particularly productive to discuss such testing in the abstract, since 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.

For automatic detection of document structures, the evaluation typically focuses on sentence breaking and sentence type detection. Since the definitions of these two types of document structures are straightforward, a standard evaluation database can be easily established.

In the basic level, the evaluation for the text normalization component should include large regression test databases of text micro-entities: addresses, Internet and email entities, numbers in many formats (ordinal, cardinal, mathematical, phone, currency, etc.), titles, and abbreviations in a variety of contexts. These would be paired with the correct reference forms in something like the SNOR used in ASR output evaluation. In its simplest form, this would consist of a database of automatically checkable paired entries like 7% vs. *seven percent*, and $1.20 vs. *one dollar and twenty cents*. If you want to evaluate the semantic capability of text normalization, the regression database might include markups for semantic tags, so that we have 7% vs. "<number>SEVEN<ratio>PERCENT</ratio></number>", and $1.20 vs. "<money>ONE DOLLAR AND TWENTY CENTS</money>". The regression database could include domain-specific entries. This implies some dependence on the system's API—its markup capabilities or tag set. In the examples given in Table 14.13, the first one is a desirable output for domain-independent input, while the second one is suitable for normalization of the same expression in mathematical formula domain.

**Table 14.13** Two examples to test domain independent/dependent text normalization.

| | |
|---|---|
| 3-4 | *three to four* |
| | *three four* |
| <math_exp> 3-4 </math_exp> | *three minus four* |

Some systems may not have a discrete level of orthographic or SNOR representation that easily lends itself to the type of evaluation described in this section. Such systems may have to evaluate their text normalization component in terms of LTS conversion.

An automated test framework for the LTS conversion analysis minimally includes a set of test words and their phonetic transcriptions for automated lookup and comparison tests. The problem is the infinite nature of language: there are always *new words* that the system does not convert correctly, and many of these will initially lack a transcription of record even to allow systematic checking. Therefore, a comprehensive test program for test of phonetic conversion accuracy needs to be paired with a data development effort. The data effort has two goals: to secure a continuous source of potential new words, such as a 24-hour newswire feed, and to maintain and construct an offline test dictionary, where reference transcriptions for new words are constantly created and maintained by human experts. This requirement illustrates the codependence of automated and manual aspects of evaluation. Different types and sources of vocabulary need to be considered separately, and they may have differing testing requirements, depending, again, on the nature of the particular system to be evaluated. For example, some systems have elaborate subsystems targeted specifically for name conversion. Such systems may depend on other kinds of preprocessing technologies, such as name identification modules, that might be tested independently.

The correct phonetic representation of a word usually depends on its sentence and even discourse contexts, as described in Section 14.6. Therefore, the adequacy of LTS conversion should not, in principle, be evaluated on the basis of isolated word pronunciations. However, a list of isolated word pronunciations is often used in LTS conversion because of its simplicity. Discourse contexts are, in general, difficult to represent unless specific applications and markup tags are available to the evaluation database. A reasonable compromise is to use a list of independent sentences with their corresponding phonetic representation for the evaluation of grapheme-to-phoneme conversion.

Error analysis should be treated as equally important as the evaluation itself. For example, if a confusability matrix shows that a given system frequently confuses central and schwa-like unstressed vowels, this may be viewed as less serious than other kinds of errors. Other subareas of LTS conversion that could be singled out for special diagnosis and testing include morphological analysis and stress placement. Of course, testing with phonemic transcriptions is the ultimate *unit test* in the sense that it contains nothing to insure that the correctly transcribed words, when spoken by the system's artificial voice and prosody, are intelligible or pleasant to hear. Phone transcription accuracy is, thus, a necessary but not a sufficient condition of quality.

## 14.10. Case Study: Festival

The University of Edinburgh's *Festival* [3] has been designed to take advantage of modular subcomponents for various standard functions. Festival provides a complete text and phonetic analysis with modules organized in sequence roughly equivalent to Figure 14.1. Festival outputs speech of quality comparable to many commercial synthesizers. While default

routines are provided for each stage of processing, the system is architecturally designed to accept alternative routines in modular fashion, as long as the data transfer protocols are followed. This variant of the traditional TTS architecture is particularly attractive for commercial purposes (development, maintenance, testing, scalability) as well as research. Festival can be called in various ways with a variety of switches and filters, set from a variety of sanctioned programming and scripting languages. These control options are beyond the scope of this overview.

### 14.10.1. Lexicon

Festival employs phonemes as the basic sounding units, which are used not only as the atoms of word transcriptions in the lexicons, but also as the organizing principle for unit selection (see Chapter 16) in the synthesizer itself. Festival can support a number of distinct phone sets and it supports mapping from one to another. A phone defined in a set can have various associated phonological features, such as vowel, high, low, etc.

The Festival lexicon, which may contain several components, provides pronunciations for words. The *addenda* is an optional list of words that are unique to a particular user, document, or application. The addenda is searched linearly. The main system lexicon is expected to be large enough to require compression and is assumed to reside on a disk or other external storage. It is accessed via binary search. The lexical entry also contains POS information, which can be modified according to the preference of the system configurer. A typical lexical entry consists of the word key, a POS tag, and phonetic pronunciation (with stress and possible syllabification indicated in parentheses):

("walkers" N ((( w ao ) 1) (( k er z ) 0)) )

If the syllables structure is not shown with parentheses, a syllabification rule component can be invoked. Separate entry lines are used for words with multiple pronunciations and/or POS, which can be resolved by later processing.

### 14.10.2. Text Analysis

Festival has been partially integrated with research on the use of automatic identification of document and discourse structures. The discourse tagging is done by a separate component, called SOLE [11]. The tags produced by SOLE indicate features that may have relevance for pitch contour and phrasing in later stages of synthesis (see Chapter 15). These must be recognized and partially interpreted at the text analysis phrase. The SOLE tags tell Festival when the text is comparing or contrasting two objects, when it's referring to old or new information, when it's using a parenthetical or starting a new paragraph, etc., and Festival will decide, based on this information, that it needs to pause, to emphasize or deemphasize, to modify its pitch range, etc.

Additionally, as discussed in Section 14.3, when document creators have knowledge about the structure or content of documents, they can express the knowledge through an XML-based synthesis markup language. A document to be spoken is first analyzed for all such tags, which can indicate alternative pronunciations, semantic or quasi-semantic attributes (different uses of numbers by context for example), as well as document structures, such as explicit sentence or paragraph divisions. The kinds of information potentially supplied by the SABLE tags[7] are exemplified in Figure 14.7.

```
<SABLE>
    <SPEAKER NAME="male1">
    The boy saw the girl in the park <BREAK/> with the telescope.
    The boy saw the girl <BREAK/> in the park with the telescope.

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

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

**Figure 14.7** A document fragment augmented with SABLE tags can be processed by the Festival system [3].

For untagged input, or for input inadequately tagged for text division (<BREAK/>), sentence breaking is performed by heuristics, similar to Algorithm 14.1, which observe whitespace, punctuation, and capitalization. A linguistic unit roughly equivalent to a sentence is created by the system for the subsequent stages of processing.

Tokenization is performed by system or user-supplied routines. The basic function is to recognize potentially speakable items and to strip irrelevant whitespace or other non-speakable text features. Note that some punctuation is retained as a feature on its nearest word.

Text normalization is implemented by token-to-word rules, which return a standard orthographic form that can, in turn, be input to the phonetic analysis module. The token-to-word rules have to deal with text normalization issues similar to those presented in Section 14.4. As part of this process, token-type-specific rule sets may be applied to disambiguate tokens whose pronunciations are highly context dependent. For example, a disambiguation routine may be required to examine context for deciding whether *St.* should be realized as *Saint* or *street*. For general English-language phenomena, such as numbers and various

[7] SABLE and other TTS markup systems are discussed further in Chapter 15.

symbols, a standard token-to-word routine is provided. One interesting feature of the Festival system is a utility for helping to automatically construct decision trees to serve text normalization rules, when system integrators can gather some labeled training data.

The linguistic analysis module for the Festival system is mainly a POS analyzer. An *n*-gram based trainable POS tagger is used to predict the likelihoods of POS tags from a limited set given an input sentence. The system uses both a priori probabilities of tags given a word and *n*-grams for sequences of tags. The basic underlying technology is similar to the work in [6] and is described in Section 14.5. When lexical lookup occurs, the predicted most likely POS tag for a given word is input with the word orthography, as a compound lookup key. Thus, the POS tag acts as a secondary selection mechanism for the several hundred words whose pronunciation may differ by POS categories.

### 14.10.3. Phonetic Analysis

The homograph disambiguation is mainly resolved by POS tags. When lexical lookup occurs, the predicted most likely POS tag for a given word is input with the word orthography as a compound lookup key. Thus, the POS tag acts as a secondary selection mechanism for the several hundred words whose pronunciation may differ by POS categories.

If a word fails lexical lookup, LTS rules may be invoked. These rules may be created by hand, formatted as shown below:

```
( # [ c h ] C = /k/ )      // ch at word start, followed by a consonant, is /k/, e.g.,
Chris
```

Alternatively, LTS rules may be constructed by automatic statistical methods, much as described in Section 14.8 above, where CART LTS systems were introduced. Utility routines are provided to assist in using a system lexicon as a training database for CART rule construction.

In addition, Festival system employs *post-lexical rules* to handle *context coarticulation*. Context coarticulation occurs when surrounding words and sounds, as well as speech style, affect the final form of pronunciation of a particular phoneme. Examples include reduction of consonants and vowels, phrase final devoicing, and *r*-insertion. Some coarticulation rules are provided for these processes, and users may also write additional rules.

## 14.11. Historical Perspective and Further Reading

Text-to-speech has a long and rich history. You can hear samples and review almost a century's worth of work at the Smithsonian's Speech Synthesis History Project [19]. A good source for multilingual samples of various TTS engines is [20].

The most influential single published work on TTS has been *From Text to Speech: The MITalk System* [1]. This book describes the MITalk system, from which a large number

of research and commercial systems were derived during the 1980s, including the widely used DECTalk system [9]. The best compact overall historical survey is Klatt's *Review of Text-to-Speech Conversion for English* [15]. For deeper coverage of more recent architectures, refer to [7]. For an overview of some of the most promising current approaches and pressing issues in all areas of TTS and synthesis, see [30]. One of the biggest upcoming issues in TTS text processing is the architectural relation of specialized TTS text processing as opposed to general-purpose natural language or document structure analysis. One of the most elaborate and interesting TTS-specific architectures is the multilingual text processing engine described in [27]. This represents a commitment to providing exactly the necessary and sufficient processing that speech synthesis requires, when a general-purpose language processor is unavailable.

However, it is expected that natural language and document analysis technology will become more widespread and important for a variety of other applications. To get an idea of what capabilities the natural language analysis engines of the future may incorporate, refer to [12] or [2]. Such generalized engines would serve a variety of clients, including TTS, speech recognition, information retrieval, machine translation, and other services which may seem exotic and isolated now but will increasingly share core functionality. This convergence of NL services can be seen in a primitive form today in Japanese *input method editors* (IME), which offload many NL analysis tasks from individual applications, such as word processors and spreadsheets, and unify these functions in a single common processor [18].

For letter-to-sound rules, NETalk [25], which describes automatic learning of LTS processes via neural network, was highly influential. Now, however, most systems have converged on decision-tree systems similar to those described in [14].

## REFERENCES

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

[2] Alshawi, H., *The Core Language Engine*, 1992, Cambridge, US, MIT Press.

[3] Black, A.W., P. Taylor, and R. Caley, "The Architecture of the Festival Speech Synthesis System," *3rd ESCA Workshop on Speech Synthesis*, 1998, Jenolan Caves, Australia, University of Edinburgh, pp. 147-151.

[4] Boguraev, B. and E.J. Briscoe, *Computational Lexicography for Natural Language Processing*, 1989, London, Longmans.

[5] Chomsky, N. and M. Halle, *The Sound Patterns of English*, 1968, Cambridge, MIT Press.

[6] Church, K., "A Stochastic Parts Program and Noun Phrase Parser for Unrestricted Text," *Proc. of the Second Conf. on Applied Natural Language Processing*, 1988, Austin, Texas, pp. 136-143.

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

[8] Francis, W. and H. Kucera, *Frequency Analysis of English Usage*, 1982, New York, N.Y., Houghton Mifflin.

[9] Hallahan, W.I., "DECtalk Software: Text-to-Speech Technology and Implementation," *Digit Technical Journal*, 1995, 7(4), pp. 5-19.

[10] Higgins, J., *Homographs*, 2000, http://www.stir.ac.uk/celt/staff/higdox/wordlist/homogrph.htm.

[11] Hitzeman, J., *et al.*, "On the Use of Automatically Generated Discourse-Level Information in a Concept-to-Speech Synthesis System," *Proc. of the Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia, pp. 2763-2766.

[12] Jensen, K., G. Heidorn, and S. Richardson, *Natural Language Processing: the PLNLP Approach*, 1993, Boston, MASS, Kluwer Academic Publishers.

[13] Jiang, L., H.W. Hon, and X. Huang, "Improvements on a Trainable Letter-to-Sound Converter," *Proc. of Eurospeech*, 1997, Rhodes, Greece, pp. 605-608.

[14] Jiang, L., H.W. Hon, and X.D. Huang, "Improvements on a Trainable Letter-to-Sound Converter," *Eurospeech'97*, 1997, Rhodes, Greece.

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

[16] LDC, *Linguistic Data Consortium*, 2000, http://www.ldc.upenn.edu/ldc/noframe.html.

[17] Levine, J., Mason, T., Brown, D., *Lex and Yacc*, 1992, Sebastopol, CA, O'Rielly & Associates.

[18] Lunde, K., *CJKV Information Processing Chinese, Japanese, Korean & Vietnamese Computing*, 1998, O'Reilly.

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

[20] Möhler, G., *Examples of Synthesized Speech*, 1999, http://www.ims.uni-stuttgart.de/phonetik/gregor/synthspeech/.

[21] Nye, P.W., *et al.*, "A Plan for the Field Evaluation of an Automated Reading System for the Blind," *IEEE Trans. on Audio and Electroacoustics*, 1973, **21**, pp. 265-268.

[22] OMF, *CML - Chemical Markup Language*, 1999, http://www.xml-cml.org/.

[23] Richardson, S.D., W.B. Dolan, and L. Vanderwende, "MindNet: Acquiring and Structuring Semantic Information from Text," *ACL'98: 36th Annual Meeting of the Assoc. for Computational Linguistics and 17th Int. Conf. on Computational Linguistics*, 1998, pp. 1098-1102.

[24] Sable, *The Draft Specification for Sable version 0.2*, 1998, http://www.cstr.ed.ac.uk/projects/sable_spec2.html.

[25] Sejnowski, T.J. and C.R. Rosenberg, *NETtalk: A Parallel Network that Learns to Read Aloud*, 1986, Johns Hopkins University.

[26] Sluijter, A.M.C. and J.M.B. Terken, "Beyond Sentence Prosody: Paragraph Intonation in Dutch," *Phonetica*, 1993, **50**, pp. 180-188.

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

[28] Sproat, R. and J. Olive, "An Approach to Text-to-Speech Synthesis," in *Speech Coding and Synthesis*, W.B. Kleijn and K.K. Paliwal, eds. 1995, Amsterdam, pp. 611-634, Elsevier Science.

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

[30] van Santen, J., *et al.*, *Progress in Speech Synthesis*, 1997, New York, Springer-Verlag.

[31] van Santen, J., *et al.*, "Report on the Third ESCA TTS Workshop Evaluation Procedure," *Third ESCA Workshop on Speech Synthesis*, 1998, Sydney, Australia.

[32] Vitale, T., "An Algorithm for High Accuracy Name Pronunciation by Parametric Speech Synthesizer," *Computational Linguistics*, 1991, **17**(3), pp. 257-276.

[33] W3C, *Aural Cascading Style Sheets (ACSS)*, 1997, http://www.w3.org/TR/WD-acss-970328.

[34] W3C, *W3C's Math Home Page*, 1998, http://www.w3.org/Math/.

[35] W3C, *Extensible Markup Language (XML)*, 1999, http://www.w3.org/XML/.

[36] Wall, L., Christiansen, T., Schwartz, R., *Programming Perl*, 1996, Sebastopol, CA, O'Rielly & Associates.

# CHAPTER 15

# Prosody

It isn't **what** you said; it's **how** you said it! Sheridan pointed out the importance of prosody more than 200 years ago [53]:

> *Children are taught to read sentences, which they do not understand; and as it is impossible to lay the emphasis right, without perfectly comprehending the meaning of what one reads, they get a habit either of reading in a monotone, or if they attempt to distinguish one word from the rest, as the emphasis falls at random, the sense is usually perverted, or changed into nonsense.*

Prosody is a complex weave of physical, phonetic effects that is being employed to express attitude, assumptions, and attention as a parallel channel in our daily speech communication. The semantic content of a spoken or written message is referred to as its *denotation*, while the emotional and attentional effects intended by the speaker or inferred by a listener are part of the message's *connotation*. Prosody has an important supporting role in guiding a listener's recovery of the basic messages (denotation) and a starring role in signaling connotation, or the speaker's attitude toward the message, toward the listener(s),

ing connotation, or the speaker's attitude toward the message, toward the listener(s), and toward the whole communication event.

From the listener's point of view, prosody consists of systematic perception and recovery of a speaker's intentions based on:

- **Pauses**: to indicate phrases and to avoid running out of air.
- **Pitch**: rate of vocal-fold cycling (fundamental frequency or F0) as a function of time.
- **Rate/relative duration**: phoneme durations, timing, and rhythm.
- **Loudness**: relative amplitude/volume.

Pitch is the most expressive of the prosodic phenomena. As we speak, we systematically vary our fundamental frequency to express our feelings about what we are saying, or to direct the listener's attention to especially important aspects of our spoken message. If a paragraph is spoken on a constant, uniform pitch with no pauses, or with uniform pauses between words, it sounds highly unnatural.

In some languages, the pitch variation is partly constrained by lexical and syntactic conventions. For example, Japanese words usually exhibit a sharp pitch fall at a certain vowel on a consistent, word-specific basis. In Mandarin Chinese [52], word meaning depends crucially on shape and register distinctions among four highly stylized syllable pitch contour types. This is a grammatical and lexical use of pitch. However, every language, and especially English, allows some range of pitch variation that can be exploited for emotive and attentional purposes. While this chapter concentrates primarily on American English, the use of some prosodic effects to indicate emotion, mood, and attention is probably universal, even in languages that also make use of pitch for signaling word identity, such as Chinese. It is tempting to speculate that speakers of some languages use expressive and affective lexical particles and interjections to express some of the same emotive effects for which American English speakers typically rely on prosody.

We discuss pausing, pitch generation, and duration separately, because it is convenient to separate them when building systems. Bear in mind, however, that all the prosodic qualities are highly correlated in human speech production. The effect of loudness is not nearly as important in synthesizing speech as the effect of the other two factors and thus is not discussed here. In addition, for many concatenative systems this is generally embedded in the speech segment.

## 15.1. THE ROLE OF UNDERSTANDING

To date, most work on prosody for TTS has focused exclusively on the utterance, which is the literal content of the message. That is, a TTS system learns whatever it can from the isolated, textual representation of a single sentence or phrase to aid in prosodic generation. Typically a TTS system may rely on word identity, word part-of-speech, punctuation, length of a sentence or phrase, and other superficial characteristics. As more sophisticated NLP

capabilities are deployed for use by TTS systems, deeper properties of an utterance, including its document or discourse context, can be taken into account.

Good prosody depends on a speaker or reader's understanding of the text's or message's meaning. As noted in [64], the *golden rule* of the Roman orator Quintilian (c. A.D. 90) states [32] "*That to make a man speak well, and pronounce with a right emphasis, he ought thoroughly to understand all that he says, be fully persuaded of it, and bring himself to have, those affections which he desires to infuse in others.*" This is clearly a tall order for today's computers! How important is understanding of the text's meaning, in generation of appropriately engaging prosody? Consider a stanza from Lewis Carroll's nonsense poem "Jabberwocky" [10]:

*Twas brillig, and the slithy toves*
*Did gyre and gimble in the wabe;*
*All mimsy were the borogoves,*
*And the mome raths outgrabe.*

Here, a full interpretation is not possible, owing primarily to lexical uncertainty (our ignorance of the meaning of words like *brillig*). However, you can recover a great deal of information from this passage that aids prosodic rendition. Foremost is probably the metrical structure of the poetic meter. This imposes a rhythmic constraint on prosodic phrasing (cadence, timing, and pause placement). Second, the function words such as *and*, *the*, *in*, etc. are interpretable and give rich clues about the general type and direction of action being specified. They also give us contextual hints about the part-of-speech of the neighboring nonsense words, which is a first crude step in interpreting those words' meaning. Third, punctuation is also important in this case. Using these three properties, with some analogical guesses about LTS conversions and stress locations in the nonsense words, would allow most speakers of English to render fairly pleasant and appropriate prosody for this poem.

Can a computer do the same? Where will a computer fall short of a human's performance on this task, and why? First, the carrier voice quality of a human reader is generally superior to synthesized voices. The natural human voice is more pleasant to a listener, all else being equal. As for the prosody per se, most TTS systems today use a fairly simple method to derive prosody, based on a distinction between closed-class function words, such as determiners and prepositions, which are thought to receive lesser emphasis, and open-ended sets of content words such as nouns like *wabe*, which are more likely to be accented. For this nonsense poem, that is essentially what most human readers do. Thus, if accurate LTS conversions are supplied, including main stress locations, a TTS system with a good synthetic voice and a reasonable default pitch algorithm of this type could probably render this stanza fairly well. Again, though the computer does not recognize it explicitly, the constrained rhythmic structure of the poem may be assisting.

But listeners to nonsense poems are generally not fully participating in the unconscious interpretive dialog, the attempt on the part of the listener to actively construct useful meaning from prosodic and message-content cues supplied in good faith by the speaker. Therefore, judgments of the prosodic quality of uninterpretable nonsense materials must always be suspect. In ordinary prose, the definition and recovery of *meaning* remains a slip-

pery question. Consider the passage below [56], which is not metrically structured, has few or no true nonsense words, and, yet, was deliberately constructed to be essentially meaningless.

> *In mathematical terms, Derrida's observation relates to the invariance of the Einstein field equation under nonlinear space-time diffeomorphisms (self-mappings of the space-time manifold which are infinitely differentiable but not necessarily analytic). The key point is that this invariance group 'acts transitively': this means that any space-time point, if it exists at all, can be transformed into any other. In this way the infinite-dimensional invariance group erodes the distinction between observer and observed; the pi of Euclid and the G of Newton, formerly thought to be constant and universal, are now perceived in their ineluctable historicity; and the putative observer becomes fatally decentered, disconnected from any epistemic link to a space-time point that can no longer be defined by geometry alone.*

Should the fact that, say, a professional news broadcaster with no prior knowledge of the author's intent could render this supposedly meaningless passage rather well, make us suspicious of any claims regarding the necessity of deep semantic analysis for high-quality prosody? Though perhaps meaningless when taken as a whole, once again, the educated human reader can certainly recover fragments of meaning from this text sufficient to support reasonable prosody. The morphology and syntax is all standard English, which takes us a long way. The quality of the announcer's rendition degrades somewhat under the condition the computer truly faces, which can be simulated by replacing the *content words* of a sentence above with content words randomly chosen from "Jabberwocky":

> *In brillig toves, Derrida's wabe gimbles to the bandersnatch of the Tumtum whiffling raths under frumious slithy diffeomorphisms (borogoves of the mimsy mome which are beamishly vorpal but not frabjously uffish).*

It is likely the human reader can still outperform the computer by reliance on morphological and syntactic cues, such as the parallelism determining the accent placements in the contrastive structure "*...which ARE...but NOT ...*" Nevertheless, the degree of *understanding* of a message's content that is required for convincing prosodic rendition remains a subtle question. Clearly, the more the machine or human reader knows, the better the prosodic rendition, but some of the most important knowledge is surprisingly shallow and accessible.

There is no rigorous specification or definition of meaning. The meaning of the rendition event itself is more significant than the inherent meaning of the text, if any. The meaning of the rendition event is determined primarily by the goals of the speaker and listener(s). While textual attributes such as metrical conventions, syntax, morphology, lexical semantics, topic, etc., contribute to the construction of both kinds of meaning, the meaning of the rendition event incorporates more important pragmatic and contextual elements, such as goals of the communication event, and speaker identity and attitude projection. Thus the concept-to-speech discussed in Chapter 17 has a much better chance of generating good prosody, since the content of the sentence is known by the SLU system.

## 15.2. PROSODY GENERATION SCHEMATIC

Figure 15.1 shows schematically the elements of prosodic generation in TTS, from pragmatic abstraction to phonetic realization. The input of the prosody module in Figure 15.1 is parsed text with a phoneme string, and the output specifies the duration of each phoneme and the pitch contour. One possible output representation of that output prosody is shown in Figure 15.2 for the sentence *The cat sat*. Up to four points per phoneme were included in this example. Often one point per phoneme is more than sufficient, except for words like *john*, where two points are needed for the phoneme */aol* to achieve a natural prosody.

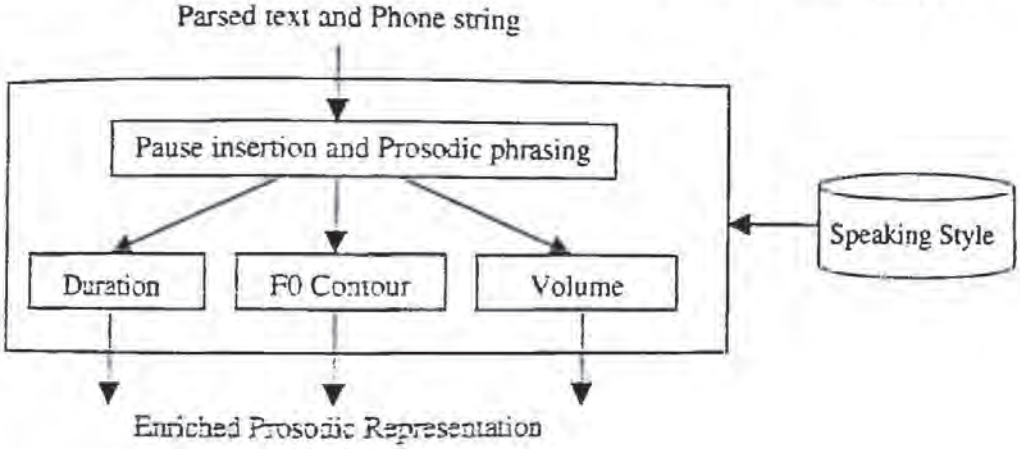



Figure 15.1 Block diagram of a prosody generation system: different prosodic representations are obtained depending on the speaking style we use.

```
DE,  24                 0,278 ;
AE0, 206
#;
I.   81                 25,278)     50,256)     (75,202);
AE.  152                1,226       25,223      (50,206)    (75,193);
T.   61                 1,178       25,178      (50,176)    (75,172);
#;
S.   206                1,278       25,272      (50,280)    (75,289);
AE.  206                1,288       25,288      (50,288)    (75,287);
T.   ...                1,220       200,220);
#;
```

Figure 15.2 Enriched prosody representation, where each line specifies the phoneme duration using the phoneme identity, the phoneme duration in milliseconds, and a number of pitch points specifying pitch and possibly volume. Each pitch point is represented by a time point, expressed as a percentage of the phoneme's duration, and the corresponding pitch value in Hz. The symbol # is a word delimiter. For example, the fourth line specifies a phoneme /t/ which lasts 85 ms and has three pitch points: the first is located at 25% of the phoneme duration, at 278 Hz, the second at 50% of the phoneme duration, at 256 Hz, and the third at the end of the phoneme, at 202 Hz. The last line specifies a silence of 200 ms, during which the pitch remains constant.

In the next sections we describe the modules of Figure 15.1: the speaking style, symbolic prosody (including pause insertion), duration assignment, and pitch generation in that order, as usually followed by most TTS systems.

## 15.3. SPEAKING STYLE

Prosody depends not only on the linguistic content of a sentence. Different people generate different prosody for the same sentence. Even the same person generates a different prosody depending on his or her mood. The *speaking style* of the voice in Figure 15.1 can impart an overall tone to a communication. Examples of such global settings include a low register, voice quality (falsetto, creaky, breathy, etc.), narrowed pitch range indicating boredom, depression, or controlled anger, as well as more local effects, such as notable excursion of pitch, higher or lower than surrounding syllables, for a syllable in a word chosen for special emphasis. Another example of a global effect is a very fast speaking rate that might signal excitement, while an example of a local effect would be the typical short, extreme rise in pitch on the last syllable of a *yes-no* question in American English.

### 15.3.1. Character

Character, as a determining element in prosody, refers primarily to long-term, stable, extralinguistic properties of a speaker, such as membership in a group and individual personality. It also includes sociosyncratic features such as a speaker's region and economic status, to the degree that these influence characteristic speech patterns. In addition, idiosyncratic features such as gender, age, speech defects, etc., affect speech, and physical status may also be a background determiner of prosodic character. Finally, character may sometimes include temporary conditions such as fatigue, inebriation, talking with mouth full, etc. Since many of these elements have implications for both the prosodic and voice quality of speech output, they can be very challenging to model jointly in a TTS system. The current state of the art is insufficient to convincingly render most combinations of the character features listed above.

### 15.3.2. Emotion

Temporary emotional conditions such as amusement, anger, contempt, grief, sympathy, suspicion, etc. have an effect on prosody. Just as a film director explains the emotional context of a scene to her actors to motivate their most convincing performance, so TTS systems need to provide information on the simulated speaker's state of mind. These are relatively unstable properties, somewhat independent of character as defined above. That is, one could imagine a speaker with any combination of social/dialect/gender/age characteristics being in any of a number of emotional states that have been found to have prosodic correlates, such as anger, grief, happiness, etc. Emotion in speech is actually an important area for future research. A large number of high-level factors go into determining emotional effects in speech. Among these are point of view (can the listener interpret what the speaker is really feeling or expressing?); spontaneous vs. symbolic (e.g., acted emotion vs. real feeling); cul-

ture-specific vs. universal; basic emotions and compositional emotions that combine basic feelings and effects; and strength or intensity of emotion. We can draw a few preliminary conclusions from existing research on emotion in speech [34]:

- Speakers vary in their ability to express emotive meaning vocally in controlled situations.
- Listeners vary in their ability to recognize and interpret emotions from recorded speech.
- Some emotions are more readily expressed and identified than others.
- Similar intensity of two emotions can lead to confusing one with the other.

An additional complication in expressing emotion is that the phonetic correlates appear not to be limited to the major prosodic variables (F0, duration, energy) alone. Besides these, phonetic effects in the voice such as jitter (inter-pitch-period microvariation), or the mode of excitation may be important [24]. In a formant synthesizer supported by extremely sophisticated controls [59], and with sufficient data for automatic learning, such voice effects might be simulated. In a typical time-domain synthesizer (see Chapter 16), the lower-level phonetic details are not directly accessible, and only F0, duration, and energy are available.

Some basic emotions that have been studied in speech include:

- **Anger**, though well studied in the literature, may be too broad a category for coherent analysis. One could imagine a threatening kind of anger with a tightly controlled F0, low in the range and near monotone; while a more overtly expressive type of tantrum could be correlated with a wide, raised pitch range.
- **Joy** is generally correlated with increase in pitch and pitch range, with increase in speech rate. Smiling generally raises F0 and formant frequencies and can be well identified by untrained listeners.
- **Sadness** generally has normal or lower than normal pitch realized in a narrow range, with a slow rate and tempo. It may also be characterized by slurred pronunciation and irregular rhythm.
- **Fear** is characterized by high pitch in a wide range, variable rate, precise pronunciation, and irregular voicing (perhaps due to disturbed respiratory pattern).

## 15.4. SYMBOLIC PROSODY

Abstract or *symbolic prosodic* structure is the link between the infinite multiplicity of pragmatic, semantic, and syntactic features of an utterance and the relatively limited F0, phone durations, energy, and voice quality. The output of the prosody module of Figure 15.2 is a

set of real values of F0 over time and real values for phoneme durations. Symbolic prosody deals with:

- Breaking the sentence into prosodic phrases, possibly separated by pauses, and
- Assigning labels, such as emphasis, to different syllables or words within each prosodic phrase.

Words are normally spoken continuously, unless there are specific linguistic reasons to signal a discontinuity. The term *juncture* refers to prosodic phrasing—that is, where do words cohere, and where do prosodic breaks (pauses and/or special pitch movements) occur. Juncture effects, expressing the degree of cohesion or discontinuity between adjacent words, are determined by physiology (running out of breath), phonetics, syntax, semantics, and pragmatics. The primary phonetic means of signaling juncture are:

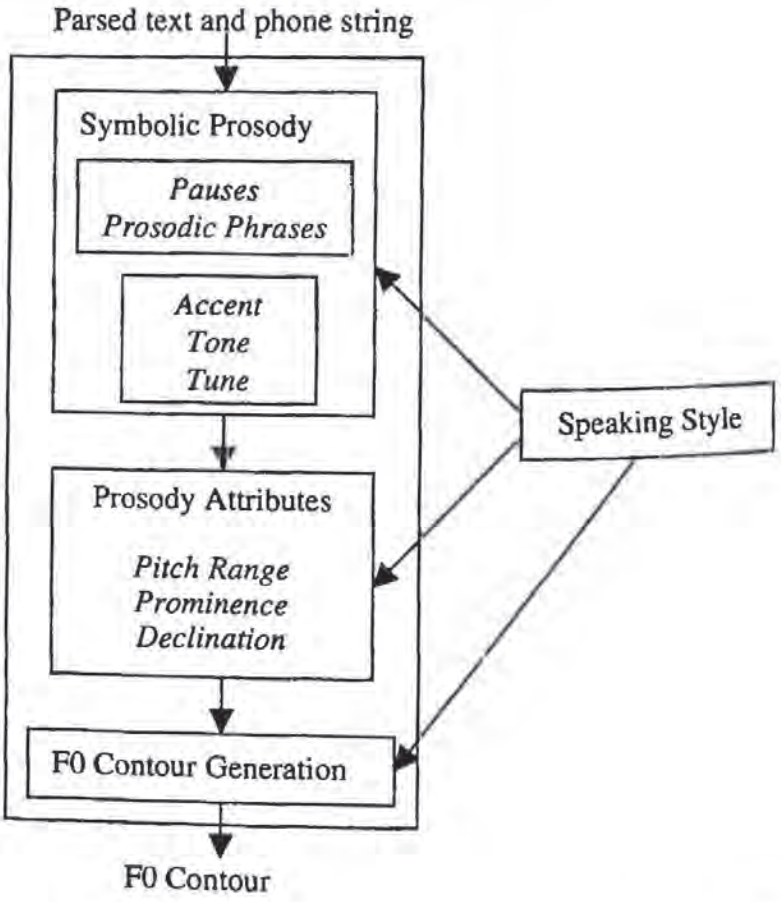



**Figure 15.3** Pitch generation decomposed in symbolic and phonetic prosody.

- Silence insertion. This is discussed in Section 15.4.1.
- Characteristic pitch movements in the phrase-final syllable. This is discussed in Section 15.4.4.
- Lengthening of a few phones in the phrase-final syllable. This is discussed in Section 15.5.
- Irregular voice quality such as vocal fry. This is discussed in Chapter 16.

Abstract prosodic structure or annotation typically specifies all the elements in the top block of the pitch-generation schematic in Figure 15.3, including *accents* (corresponding conceptually to *heads* in standard syntactic structure). The accent types are selected from a small inventory of *tones* for American English (e.g., high, low, rising, late-rising, scooped). The sequence of accent and juncture tones in a given prosodic structure may cohere to yield *tune*-like effects that have some holistic semantic interpretation. While we center our description in an abstract representation called ToBI, we also describe other alternate representations in Section 15.4.6. Finally, though in principle the prosody attributes module applies to all prosody variables, it is mostly used for F0 generation in practice, and as such is discussed in Section 15.6.1.

### 15.4.1. Pauses

In a long sentence, speakers normally and naturally pause a number of times. These pauses have traditionally been thought to correlate with syntactic structure but might more properly be thought of as markers of information structure [58]. They may also be motivated by poorly understood stylistic idiosyncrasies of the speaker, or physical constraints. In spontaneous speech, there is also the possibility that some pauses serve no linguistic function but are merely artifacts of hesitation.

In a typical system, the most reliable indicator of pause location is punctuation. After resolution of abbreviations and special symbols relevant to text normalization (Chapter 14), the remaining punctuation can be reclassified as essentially prosodic in nature. This includes periods, commas, exclamation points, parentheses, ellipsis points, colons, dashes, etc. Each of these can be taken to correspond to a prosodic phrase boundary and can be given a special pitch movement at its end-point.

In predicting pauses, although you have to consider both their occurrence and their duration, the simple presence or absence of a silence (of greater than 30 ms) is the most significant decision, and its exact duration is secondary, based partially on the current rate setting and other extraneous factors.

There are many reasonable places to pause in a long sentence, but a few where it is critical *not* to pause. The goal of a TTS system should be to avoid placing pauses anywhere that might lead to ambiguity, misinterpretation, or complete breakdown of understanding. Fortunately, most decent writing (apart from email) incorporates punctuation according to exactly this metric: no need to punctuate after every word, just where it aids interpretation.

Therefore, by simply following punctuation in many writing styles, the TTS system will not go far wrong.

Consider the opening passage from Edgar Allan Poe's classic story *The Cask of Amontillado* (1846) arranged sentence-by-sentence:

1. *The thousand injuries of Fortunato I had borne as I best could, but when he ventured upon insult, I vowed revenge.*
2. *You, who so well know the nature of my soul, will not suppose, however, that I gave utterance to a threat.*
3. *At length I would be avenged; this was a point definitively settled—but the very definitiveness with which it was resolved precluded the idea of risk.*

If we place prosodic pauses at all and only the punctuation sites, the result is acceptable to most listeners, and no definite mistakes occur. Some stretches seem a bit too long, however. Perhaps the second part of sentence 3 could be broken up as follows:

*but the very definitiveness with which it was resolved PAUSE precluded the idea of risk.*

While commas are particularly useful in signaling pause breaks, as seen above, pauses may be optional following comma-delimited listed words (*berries, melons, and cheese*), though the special small pitch rise typical of a minor (nonpause) break is often present.

Cases where placing a boundary in certain locations critically affects interpretation include tag questions and verb particle constructions (where the verb must not be separated from its particle), such as:

*Why did you hit Joe?*
*Why did you hit PAUSE Joe?*

He distractedly threw out the trash.
*(NOT ... threw PAUSE out ...)*

He distractedly gazed PAUSE out the window.
*(NOT ... out PAUSE the ...)*

Supplying junctures at the optimal points sometimes requires deep semantic analysis provided by the module described in Chapter 14. The need for independent methods for pause insertion has motivated some researchers to assume that no independent source of natural language analysis is available. The CART discussed in Chapter 4 can be used for pause assignment [36]. You can use POS categories of words, punctuation, and a few structural measures, such as overall length of a phrase, and length relative to neighboring phrases to construct the classification tree. The decision-tree-based system can have correct prediction of 81% for pauses over test sentences with only 4% false prediction rates. As the algo-

rithm proceeds successively left to right through each pair of words, the following questions can be used:

- *Is this a sentence boundary (marked by punctuation)?*
- *Is the left word a content word and the right word a function word?*
- *What is the function word type of word to the right? (Certain function words are more likely to signal a break)*
- *Is either adjacent word a proper name (capitalized)?*
- *How many content words have occurred since the previous function word (If > 4 or 5 words, a break is more likely)*
- *Is there a comma at this location?*
- *What is the current location in the sentence?*
- *What is the length of current proposed major phrase?*

These questions summarize the relevant knowledge, which could be formulated in expert-system rules, and augmented by high-quality syntactic knowledge if available, or trained statistically from tagged corpora.

### 15.4.2. Prosodic Phrases

An end-of-sentence period may trigger an extreme lowering of pitch, a comma-terminated prosodic phrase may exhibit a small *continuation rise* at its end, signaling more to come, etc. Rules based on these kinds of simple observations are typically found in commercial TTS systems. Certain pitch-range effects over the entire clause or utterance can also be based on punctuation—for example, the range in a parenthetical restrictive clause is typically narrower than that of surrounding material, while exclamations may have a heightened range, or at least higher accent targets throughout.

Prosodic junctures that are clearly signaled by silence (and usually by characteristic pitch movement as well), also called *intonational phrases*, are required between utterances and usually at punctuation boundaries. Prosodic junctures that are not signaled by silence but rather by characteristic pitch movement only, also called *phonological phrases*, may be harder to place with certainty and to evaluate. In fast speech, the silence demarcating fruits in the sentence 'We have blueberries, raspberries, gooseberries, and blackberries.' may disappear, yet a trace of the continuation rise on each 'berries' typically remains. These locations would then still qualify as minor intonation phrases, or phonological phrases.

In analyzing spontaneous speech, the nature and extent of the signaling pitch movement may vary from speaker to speaker. A further consideration for practical TTS systems is a user's preferred rate setting: blind people who depend on TTS to access information in a computer usually prefer a fast rate, at which most sentence-internal pauses should disappear.

To discuss linguistically significant juncture types and pitch movement, it helps to have a simple standard vocabulary. ToBI (for *Tones and Break Indices*) [4, 55] is a proposed

standard for transcribing symbolic intonation of American English utterances, though it can be adapted to other languages as well. The *Tones* part of ToBI is considered in greater detail in Section 15.4.4.

The *Break Indices* part of ToBI specifies an inventory of numbers expressing the strength of a prosodic juncture. The Break Indices are marked for any utterance on their own discrete *break index tier* (or layer of information), with the BI notations aligned in time with a representation of the speech phonetics and pitch track. On the break index tier, the prosodic association of words in an utterance is shown by labeling the end of each word for the subjective strength of its association with the next word, on a scale from 0 (strongest perceived conjoining) to 4 (most disjoint), defined as follows:

- **0** for cases of clear phonetic marks of clitic[1] groups (phrases with appended reduced function words), e.g., the medial affricate in contractions of *did you* or a flap as in *got it*.
- **1** most phrase-medial word boundaries.
- **2** a strong disjuncture marked by a pause or virtual pause, but with no tonal marks, i.e., a well-formed tune continues across the juncture. OR, a disjuncture that is weaker than expected at what is tonally a clear intermediate or full intonation phrase boundary.
- **3** intermediate intonation phrase boundary, i.e., marked by a single phrase tone affecting the region from the last pitch accent to the boundary.
- **4** full intonation phrase boundary, i.e., marked by a final boundary tone after the last phrase tone.

For example, a typical fluent utterance of the following sentence: *Did you want an example?* might have a 0 between *Did* and *you*, indicating palatalization of the */d j/* sequence across the boundary between these words. Similarly, the break index value between *want* and *an* might again be 0, indicating deletion of */t/* and subsequent flapping of */n/*. The remaining break index values would probably be 1 between *you* and *want* and between *an* and *example*, indicating the presence of a mere word boundary, and 4 at the end of the utterance, indicating the end of a well-formed intonation phrase. The annotation is thus:

*Did-0 you-1 want-0 an-1 example-4?*

Without reference to any other knowledge, therefore, a system would place a 1 after every word, except where utterance-final punctuation motivates placement of 4. Perhaps comma boundaries would be marked by 4. Need any more be done? A BI of 0 correlates with any special phone substitution or modification rules for reduction in clitic groups that a TTS system may attempt. By marking the location of clitic (close association) phonetic reduction, such a BI can serve as a trigger for special duration rules that shorten the segments of the cliticized word. Whatever syntactic/semantic processing was done to propose the cliticization can serve to trigger assignment of 1. The 2 mark is generally more useful for

[1] Pronounced as part of another word, as in *ve* in *I've*.

analysis than for speech generation. You may observe that in the literature on intonation, a 3 break is sometimes referred to as an *intermediate phrase* break, or a *minor phrase* break, while a 4 break is sometimes called an *intonational phrase* break or a *major phrase* break.

### 15.4.3. Accent

We should briefly clarify use of terms such as stress and accent. *Stress* generally refers to an idealized location in an English word that is a potential site for phonetic prominence effects, such as extruded pitch and/or lengthened duration. This information comes from our standard lexicon. Thus, the second syllable of the word *em-ploy-er* is said to have the abstract property of lexical stress. In an actual utterance, if the word as a whole is sufficiently important, phonetic highlighting effects are likely to fall on the lexically *stressed* syllable:

*Acme **Industries** is the **biggest** **employer** in the **area**.*

*Accent* is the signaling of semantic salience by phonetic means. In American English, accent is typically realized via extruded pitch (higher or lower than the general trend) and possibly extended phone duration. Although lexical stress as noted in dictionaries is strictly an abstract property, these accent-signaling phonetic effects are usually strongest on the lexically stressed syllable of the word that is singled out for accentuation (e.g., *employer*).

In the sentence above, the word *employer* is not specially focused or contrasted, but it is an important word in the utterance, so its lexically stressed syllable typically receives a prosodic accent (via pitch/duration phonetic mechanisms), along with the other syllables in boldface. In cases of special emphasis or contrast, the lexically specified preferred location of stress in a word may be overridden in utterance accent placement:

*I didn't say employer, I said **employee**.*

It is also possible to override the primary stress of a word with the secondary stress where a neighboring word is accented. While normally we would say *Massachusetts*, we might say *Massachusetts legislature* [51].

Let's consider what might make a word accentable in context. A basic rule based on the use of POS category is to decide accentuation by *accenting all and only the content words*. Such rule is used in the baseline F0 generation system of Section 15.6.2. The content words are major open-class categories such as noun, verb, adjective, adverb, and certain strong closed-class words such as negatives and some quantifiers. Thus, the *function words*, made up of closed-class categories such as prepositions, conjunctions, etc., end up on a kind of *stop list* for accentuation, analogous to the stop lists used traditionally in simple document indexing schemes for information retrieval. This works adequately for many short, isolated sentences, such as "*The **cat** **sat** on the **mat**.*", where the words selected for accentuation appear in boldface. For more complex sentences, appearing in document or dialog context, such an algorithm will sometimes fail.

How often does the POS class-based stop-list approach fail? Let's consider a slightly more elaborate variant on the theme. A model was created using the Lancaster/IBM Spoken

English Corpus (SEC) [3]. This includes a variety of text types, including news, academic lectures, commentary, and magazine articles. Each word in the corpus has a POS tag automatically assigned by an independent process. The model predicts the probability of a word of POS having accent status. The probability is computed based on POS class of a sequence of words in the history in the similar way as $n$-gram models discussed in Chapter 11. This simple model performed at or above 90% correct predictions for all text types. As for stress predictions that were *incorrect*, we should note that in many cases accents are optional—it is more a game of avoiding plain wrong predictions than it is of finding optimal ones. Clearly, however, there are situations that call for greater power than a simple POS-based model can provide. Even different readings of the exact same text can result in different accents [46].

Consider a simple case where a word or its base form is repeated within a short paragraph. Such words may have the necessary POS to trigger accentuation, but, since they have already been mentioned (perhaps with varying morphological inflection), it can sound strange to highlight them again with accentuation. They are *given* or *old* information the second time around and may be deaccented. For example, the second occurrence of the noun 'switch' below is best not accented:

> *At the **corner** of the keyboard are **two switches**.*
> *The **top** switch is **user-defined**.*

To achieve this, the TTS system can keep a queue of most recently used words or normalized base forms (if morphological capability is present), and block accentuation when the next word has been used recently. The queue should be reset periodically, perhaps at paragraph boundaries [54].

Of course, the surface form of words, even if reduced to a base form or lemma by morphology, won't always capture the deeper semantic relations that govern accentuation. Consider the following fragment extracted from Roger Rosenblatt's essay:

> ***Kids** today are being neglected by the **older** generation. Adults spend **hours** every day on the **StairMaster**, trying to **become** the youth they should be attending to.*

A simple content-word-based accentuation algorithm accents the word *youth*, because it is a noun. In context, however, it is not optimal to accent *youth*, because it is co-referent with the subject of the fragment, which is *kids today*. Thus it is, by some metrics, old or *given* information, and it had better remain unaccented. The surrounding verbs *become*, *should*, and *attending* may get extra prominence. The degree to which coreference relations, from surface identity to deep anaphora, can be exploited depends on the power of the NL analysis supporting the TTS function.

Other confusions can arise in word accentuation due to English complex nominals, where lack of, or location of, an accent may be a lexical (static) rather than a syntactic or dynamic property. Consider:

> *I **invited** her to my **birthday** party, but she **said** she **can't attend** any parties until her **grades** improve.*

One possible accent structure is indicated in boldface. Here *birthday party* functions as a complex nominal, with lexical stress on *birthday*. The word *party* should not receive stress at all, nor should its later stand-alone form *parties*. Accentuation of *improve* is optional: it is a full content word, yet somehow it also feels predictable from the context, so deaccentuation is possible. Some of the complex nominals, like *birthday party*, are fully fixed and can be entered into the lexicon as such. Others form small families of binary or *n*-ary phrases, which may be detected by local syntactic and lexical analysis. Ambiguous cases such as *moving van* or *hot dog*, which could be either nominals or adjective-noun phrases, may have to be resolved by user markup or text understanding processes.

Dwight Bolinger opined that *Accent is predictable—if you're a mind reader* [6], asserting that accentuation algorithms will never achieve perfect performance, because a writer's exact intentions cannot be inferred from text alone, and understanding is needed. However, work in [20] (similar to [3] but incorporating more sophisticated mechanics for name identification and memory of recent accented items), showed that reasonably straightforward procedures, if applied separately and combined intelligently, can yield adequate results on the accentuation task. This research has also determined that improvement occurs when the system learns that not all 'closed-class' categories are equally likely to be deaccented. For example, *closed accented* items include the negative article, negative modals, negative *do*, most nominal pronouns, most nominative and all reflexive pronouns, pre- and postqualifiers (e.g., quite), prequantifiers (e.g., *all*), postdeterminers (e.g., *next*), nominal adverbials (e.g., *here*), interjections, particles, most wh-words, plus some prepositions (e.g., *despite, unlike*).

One area of current and future development is the introduction of discourse analysis to synthesis of dialog. Discourse analysis algorithms attempt to delimit the time within which a given word/concept can be considered newly introduced, given, old, or reintroduced, and combined with analysis of segments within discourse and their boundary cues (turn-taking, digressions, interruptions, summarization, etc.) can supplement algorithms for accent assignment. This kind of work improves the naturalness of computer responses in human-computer dialog, as well as the accentuation in TTS renditions of pure text, when dialog must be performed (e.g., in reading a novel out loud) [44].

As noted above, user- or application-supplied annotations, based on intimate knowledge of the purpose and content of the speech event, can greatly enhance the quality by offloading the task of automatic accent prediction. The /EMPHASIS/ tag described in Section 15.7, with several levels of strength including *reduced accent* and *no accent*, is ideally suited for this purpose.

### 15.4.4. Tone

*Tones* can be understood as labels for perceptually salient levels or movements of F0 on syllables. Pitch levels and movements on accented and phrase-boundary syllables can exhibit a bewildering diversity, based on the speaker's characteristics, the nature of the speech event, and the utterance itself, as discussed above. For modeling purposes, it is useful to

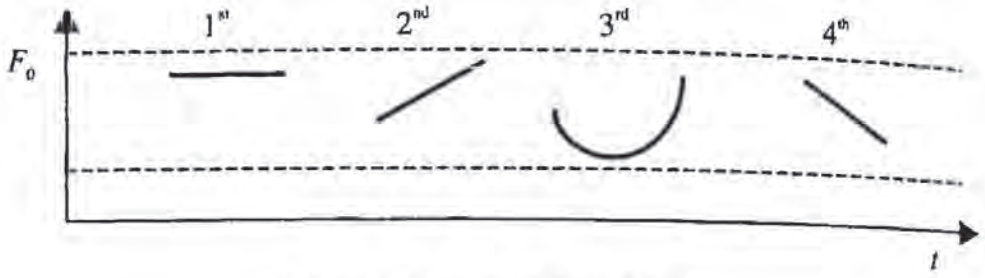



**Figure 15.4** The four Chinese tones.

have an inventory of basic, abstract pitch types that could in principle serve as the base inventory for expression of linguistically significant contrasts. Chinese, a lexical tone language, is said to have an inventory of 4 lexical tones (5 if neutral tone is included), as shown in Figure 15.4. Different speakers can realize these tones differently according to their physiology, mood, utterance content, and the speech occasion. But the variance in the tones' shapes, and contrasts with one another, remain fairly predictable, within broad limits.

By analogy, linguists have proposed a relatively small set of tonal primitives for English, which can be used, in isolation or in combination, to specify the gross phonological typology of linguistically relevant contrasts found in theories of English intonational meaning [17, 28, 39]. A basic set of tonal contrasts has been codified for American English as part of the Tones and Break Indices (ToBI) system [4, 55]. These categories can be used for annotation of prosodic training data for machine learning, and also for internal modular control of F0 generation in a TTS system. The set specifies 2 abstract levels, H(igh) and L(ow), indicating a relatively higher or lower point in a speaker's range. The H/L primitive distinctions form the foundations for 2 types of entities: pitch accents, which signal prominence or culmination; and boundary tones, which signal unit completion, or delimitation. The boundary tones are further divided into phrase types and full boundary types, which would mark the ends of intonational phrases or whole utterances.

While useful as a link to syntax/semantics, the term *accent* as defined in Section 15.4.3 is a bit too abstract, even for symbolic prosody. What is required is a way of labeling linguistically significant types of pitch contrast on accented syllables. Such a system could serve as the basis for a theory of intonational meaning. The ToBI standard specifies six types of pitch accents (see Table 15.1) in American English, where the * indicates direct alignment with an accented syllable, two intermediate phrasal tones (see Table 15.2), and five boundary tones [4] (see Table 15.3).

In American English one sometimes hears a string of strictly descending pitch accent levels across a short phrase. When judiciously applied, this *downstep* effect can be pleasantly natural, as in the following sentence:

*"I saw a big-H**

*fat-!H**

*pig-!H* (L-L%)"*

A basic rule used in the baseline F0 generation system of Section 15.6.2 consists in having all the pitch accents realized as H*, associated with the lexically stressed syllable of accented words. In general, ToBI representations of intonation should be sparse, specifying only what is linguistically significant. So, words lacking accent should not receive ToBI pitch accent annotations, and their pitch must be derived via interpolation over neighbors, or by some other default means. Low excursions can be linguistically significant also, in the crude sense that if you dip very low in your range on a given word, it may be perceived as prominent by listeners. L*+H and L+H* are both F0 rises on the accented syllable, but in the case of L*+H, the association of the starred tone (L*) with the accented syllable may push the realization of H off to the following syllable. !H* can be used for successively lowered high accents, such as might be found on *big red car*, or *tall, dark, and handsome*. A ToBI labeled utterance is shown in Figure 15.5.

A typical boundary tone is the *final lowering*, the marked tendency for the final syllable in all kinds of noninterrogative utterances to be realized on a pitch level close to the absolute bottom of a speaker's range. The final low (L-L%) may 'pull down' the height of some few accents to its left as well [41].

**Table 15.1** ToBI pitch accent tones.

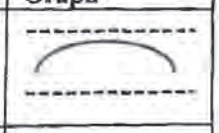
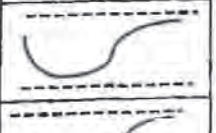
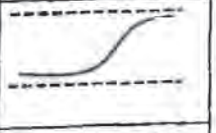
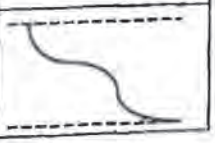
| ToBI tone | Description | Graph |
|---|---|---|
| H* | *peak accent*—a tone target on an accented syllable which is in the upper part of the speaker's pitch range. |  |
| L* | *low accent*—a tone target on an accented syllable which is in the lowest part of the speaker's pitch range |  |
| L*+H | *scooped accent*—a low tone target on an accented syllable which is immediately followed by a relatively sharp rise to a peak in the upper part of the speaker's pitch range. |  |
| L*+!H | *scooped downstep accent*—a low tone target on an accented syllable which is immediately followed by a relatively flat rise to a downstep peak. |  |
| L+H* | *rising peak accent*—a high peak target on an accented syllable which is immediately preceded by a relatively sharp rise from a valley in the lowest part of the speaker's pitch range. |  |
| !H* | *downstep high tone*—a clear step down onto an accented syllable from a high pitch which itself cannot be accounted for by an H phrasal tone ending the preceding phrase or by a preceding H pitch accent in the same phrase. |  |

**Table 15.2** ToBI intermediate phrasal tones.

| ToBI tone | Description |
|---|---|
| L- | Phrase accent, which occurs at an intermediate phrase boundary (level 3 and above). |
| H- | Phrase accent, which occurs at an intermediate phrase boundary (level 3 and above). |

Ultimately, abstract linguistic categories should correlate with, or provide labels for expressing, contrasts in meaning. While the ToBI pitch accent inventory is useful for generating a variety of English-like F0 effects, the distinction between perceptual contrast, functional contrast, and semantic contrast is particularly unclear in the case of prosody [41]. For example, whether or not the L*, an alternative method of signaling accentual prominence, functions in full linguistic contrast to H* is unclear.

In addition, we have mentioned that junctures are typically marked with perceptible pitch movements that are independent of accent. The ToBI specification also allows for combinations of the H and L primitives that signal phrase, clause, and utterance boundaries. These are called phrasal tones. The ToBI specification further points out that since intonation phrases are composed of one or more intermediate phrases plus a boundary tone, full intonation phrase boundaries have two final tones.

The symbolic ToBI transcription alone is not sufficient to generate a full F0 contour. The remaining components are discussed in Section 15.6.

**Table 15.3** ToBI boundary tones.

| ToBI tone | Description |
|---|---|
| L-L% | For a full intonation phrase with an L phrase accent ending its final intermediate phrase and an L% boundary tone falling to a point low in the speaker's range, as in the standard 'declarative' contour of American English. |
| L-H% | For a full intonation phrase with an L phrase accent closing the last intermediate phrase, followed by an H boundary tone, as in 'continuation rise.' |
| H-H% | For an intonation phrase with a final intermediate phrase ending in an H phrase accent and a subsequent H boundary tone. as in the canonical 'yes-no question' contour. Note that the H- phrase accent causes 'upstep' on the following boundary tone, so that the H% after H- rises to a very high value. |
| H- L% | For an intonation phrase in which the H phrase accent of the final intermediate phrase upsteps the L% to a value in the middle of the speaker's range, producing a final level plateau. |
| %H | High initial boundary tones; marks a phrase that begins relatively high in the speaker's pitch range when not explained by an initial H* or preceding H%. |

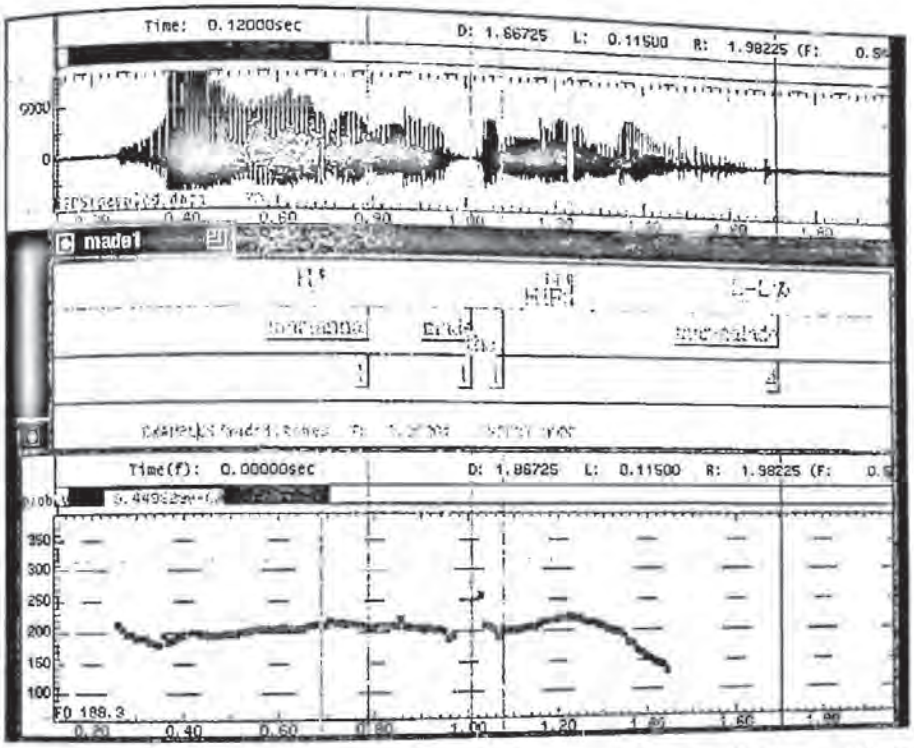



**Figure 15.5** "*Marianna made the marmalade*", with an H* accent on *Marianna* and *marmalade*, and a final L-L% marking the characteristic sentence-final pitch drop. Note the use of 1 for the weak inter-word breaks, and 4 for the sentence-final break (after Beckman [4]).

### 15.4.5. Tune

*Nyaah nuh nyaah you get*
*nyaah nyaah, can't me!*
*—Children's chant*

Some pitch contours appear to be immediately recognizable and emotionally interpretable, independent of lexical content, such as the English *children's chant* above [40]. Can this idea of stylized *tunes*, perhaps decomposable into the *tones* we examined above, be applied to the intonation of ordinary speech? In fact, the ideal use of the ToBI pitch accent labels above would be as primitive elements in holistic prosodic contour descriptions, analogous to the role of phonemes in words. Ultimately, a dictionary of meaningful contours, described abstractly by ToBI tone symbols to allow for variable phonetic realization, would constitute a theory of intonational meaning for American English. Ideally, the meanings of such contours could perhaps be derived compositionally from the meanings of their constituent pitch accent and boundary tones, thus allowing us to dispense with the dictionary altogether. Contour stylization approaches describe contours holistically and index them for

application on the basis of utterance type, usually based on a naïve syntactic typology, e.g., question, declarative, etc.

The holistic representation of contours can perhaps be defended, but the categorizing of types via syntactic description (usually triggered by punctuation) is questionable. Typically, use of punctuation as a rule trigger for pitch effects is making certain hidden assumptions about the relation between punctuation and syntax, and in turn between syntax and prosody. An obvious example is question intonation. If you find a question mark at the end of a sentence, are you justified in applying a final high rise (which might be denoted as H-H% in ToBI)? First, the intonation of yes-no questions in general differs from that of *wh*-questions. *Wh*-questions usually lack the extreme final upturn of F0 heard in some yes-no questions:

i. *Are you going?*
ii. *Where are you going?*

However, there are cases where an extreme final upturn is acceptable on ii. As [8] puts it, "It has been emphasized repeatedly ... that no intonation is an infallible clue to any sentence type: any intonation that can occur with a statement, a command, or an exclamation can also occur with a question."

Admittedly, there is a rough correspondence between syntactic types and speech acts,[2] as shown in Table 15.4. Nevertheless, the correspondence between syntactic types and acts is not deterministic, and prosody in spontaneous speech is definitely mediated via speech acts (the pragmatic context and use of an utterance) rather than syntactic types. Thus, it is difficult to obtain high-quality simulation of spontaneous speech based on linguistic descriptions that do not include speech acts and pragmatics. Likewise, even simulation of prose reading without due consideration of pragmatics and speech acts, and based solely on syntactic types, is difficult because prose that is read may:

- Include acted dialog
- Have limited occurrence of most types other than declarative, lessening variety in practice
- Include long or complex sentences, blunting 'stereotypical' effects based on utterance type
- Lack text cues as to syntactic type, or analysis grammar may be incomplete

**Table 15.4** Relationship between syntactic types and speech acts.

| Type | Speech Act | Example |
|---|---|---|
| interrogative | Questioning | Is it good? |
| declarative | Stating | It's good. |
| imperative | Commanding | Be good! |
| exclamatory | Exclaiming | How good it is! |

[2] For a more in-depth coverage of speech acts, consult Chapter 17.

Thus, description of an entire speech event, rather than inferences about text content, is again the ultimate guarantor of quality. This is why the future of automatic prosody lies with *concept-to-speech* systems (see Chapter 17) incorporating explicit pragmatic and semantic context specification to guide message rendition.

For commercial TTS systems that must infer structure from raw text, there are a few characteristic fragmentary pitch patterns that can be taken as tunes and applied to special segments of utterances. These include:

- Phone numbers—downstepping with pauses
- List intonation—downstepping with pauses (melons, pears, and eggplants)
- Tag and quotative tag intonation—low rise on tag (*Never!* he blurted. Come here, Jonathan.)

## 15.4.6. Prosodic Transcription Systems

ToBI, introduced above, can be used as a notation for transcription of prosodic training data and as a high-level specification for the symbolic phase of prosodic generation. Alternatives to ToBI also exist for these purposes, and some of them are amenable to automated prosody annotation of corpora. Some examples of this type of system are discussed in this section.

PROSPA was developed specially to meet the needs of discourse and conversation analysis, and it has also influenced the Prosody Group in the European ESPRIT 2589 SAM (Multilingual Speech Input/Output Assessment, Methodology and Standardization) project [50]. The system has annotations for general or global trends over long spans shown in Table 15.5, short, accent-lending pitch movements on particular vowels are transcribed in Table 15.6, and the pitch shape after the last accent in a () sequence, or *tail*, is indicated in Table 15.7.

**Table 15.5** Annotations for general or global trends over long spans.

| | |
|---|---|
| ( ) | extent of a sequence of cohesive accents |
| F | globally falling intonation |
| R | globally rising intonation |
| H | level intonation on high tone level |
| M | level intonation on middle tone level |
| L | level intonation on low tone level |
| H/F | falling intonation on a globally high tone level |
| ... | sequence of weakly accented or unaccented syllables |

**Table 15.6** Annotations for accent-lending pitch movements on particular vowels.

| | |
|---|---|
| + | Upward pitch movement |
| - | Downward pitch movement |
| = | level pitch accent |

**Table 15.7** Annotations for pitch shape after the last accent in a () sequence, or *tail*.

| | |
|---|---|
| | falling tails |
| / | rising tails |
| - | level tails |
| ^ | combinations of tails (rising-falling here) |

INTSINT is a coding system of intonation described in [22]. It provides a formal encoding of the symbolic or phonologically significant events on a pitch curve. Each such target point of the stylized curve is coded by a symbol, either as an absolute tone, scaled globally with respect to the speakers pitch range, or as a relative tone, defined locally in conjunction with the neighboring target points. Absolute tones in INTSINT are defined according to the speaker's pitch range as shown in Table 15.8. Relative tones are notated in INTSINT with respect to the height of the preceding and following target points, as shown in Table 15.9.

**Table 15.8** The definition of absolute tones in INTSINT.

| | |
|---|---|
| T | top of the speaker's pitch range |
| M | initial, mid value |
| B | bottom of the speaker's pitch range |

**Table 15.9** The definition of relative tones in INTSINT.

| | |
|---|---|
| H | target higher than both immediate neighbours |
| L | target lower than both immediate neighbours |
| S | target not different from preceding target |
| U | target in a rising sequence |
| D | target in a falling sequence |

In a transcription, numerical values can be retained for all F0 target points. TILT [60] is one of the most interesting models of prosodic annotation. It can represent a curve in both its qualitative (ToBI-like) and quantitative (parametrized) aspects. Generally any 'interesting' movement (potential pitch accent or boundary tone) in a syllable can be described in terms of TILT events, and this allows annotation to be done quickly by humans or machines without specific attention to linguistic/functional considerations, which are paramount for ToBI labeling. The linguistic/functional correlations of TILT events can be linked by subsequent analysis of the pragmatic, semantic, and syntactic properties of utterances.

The automatic parametrization of a pitch event on a syllable is in terms of:

- starting F0 value (Hz)
- duration
- amplitude of rise ($A_{rise}$, in Hz)
- amplitude of fall ($A_{fall}$, in Hz)
- starting point, time aligned with the signal and with the vowel onset

The tone shape, mathematically represented by its *tilt*, is a value computed directly from the F0 curve by the following formula:

$$tilt = \frac{|A_{rise}| - |A_{fall}|}{|A_{rise}| + |A_{fall}|} \tag{15.1}$$

A likely syllable for tilt analysis in the contour can be automatically detected based on high energy and relatively extreme F0 values or movements. Human annotators can select syllables for attention and label their qualities according to Table 15.10.

**Table 15.10** Label scheme for syllables.

| | |
|---|---|
| sil | Silence |
| c | Connection |
| a | Major pitch accent |
| fb | Falling boundary |
| rb | Rising boundary |
| afb | Accent+falling boundary |
| arb | Accent+rising boundary |
| m | Minor accent |
| mfb | Minor accent+falling boundary |
| mrb | Minor accent+rising boundary |
| l | Level accent |
| lrb | Level accent+rising boundary |
| lfb | Level accent+falling boundary |

## 15.5. DURATION ASSIGNMENT

Pitch and duration are not entirely independent, and many of the higher-order semantic factors that determine pitch contours may also influence durational effects. The relation between duration and pitch events is a complex and subtle area, in which only initial

exploration has been done [63]. Nonetheless, most systems often treat duration and pitch independently because of practical considerations [61].

Numerous factors, including semantics and pragmatic conditions, might ultimately influence phoneme durations. Some factors that are typically neglected include:

- The issue of speech rate relative to speaker intent, mood, and emotion.
- The use of duration and rhythm to possibly signal document structure above the level of phrase or sentence (e.g., paragraph).
- The lack of a consistent and coherent practical definition of the phone such that boundaries can be clearly located for measurement.

### 15.5.1. Rule-Based Methods

Klatt [1] identified a number of first-order perceptually significant effects that have largely been verified by subsequent research. These effects are summarized in Table 15.11.

**Table 15.11.** Perceptually significant effects for duration. After Klatt [1].

| |
|---|
| Lengthening of the final vowel and following consonants in prepausal syllables. |
| Shortening of all syllabic segments[3] in nonprepausal position. |
| Shortening of syllabic segments if not in a word final syllable. |
| Consonants in non-word-initial position are shortened. |
| Unstressed and secondary stressed phones are shortened. |
| Emphasized vowels are lengthened. |
| Vowels may be shortened or lengthened according to phonetic features of their context. |
| Consonants may be shortened in clusters. |

The rule-based duration-modeling mechanism involves table lookup of minimum and inherent durations for every phone type. The minimum duration is rate dependent, so all phones could be globally scaled in their minimum durations for faster or slower rates. The inherent duration is the raw material for the rules above: it may be stretched or contracted by a prespecified percentage attached to each rule type above applied in sequence, then it is finally added back onto the minimum duration to yield a millisecond time for a given phone. The duration of a phone is expressed as

$$d = d_{min} + r(\bar{d} - d_{min}) \tag{15.2}$$

[3] Syllabic segments include vowels and syllabic consonants.

where $d_{min}$ is the minimum duration of the phoneme, $\bar{d}$ is the average duration of the phoneme, and the correction $r$ is given by

$$r = \prod_{i=1}^{N} r_i \tag{15.3}$$

for the case of $N$ rules being applied where each rule has a correction $r_i$. At the very end, a rule may apply that lengthens vowels when they are preceded by voiceless plosives (/p/, /t/, /k/). This is also the basis for the additive-multiplicative duration model [49] that has been widely used in the field.

### 15.5.2. CART-Based Durations

A number of generic machine-learning methods have been applied to the duration assignment problem, including CART and linear regression [43, 62]. The voice datasets generally rely on less than the full set of possible joint duration predictors implied in the rule list of Table 15.11. It has been shown that a model restricted to the following features and contexts can compare favorably, in listeners' perceptions, with durations from natural speech [43]:

- Phone identity
- Primary lexical stress (binary feature)
- Left phone context (1 phone)
- Right phone context (1 phone)

In addition, a single rule of vowel and post-vocalic consonant lengthening (rule 1 in Table 15.11) is applied in prepausal syllables. The restriction of phone context to immediate left and right neighbors results in a triphone duration model, congruent with the voice triphone model underlying the basic synthesis in the system [23]. In perceptual testing this simple triphone duration model yielded judgments nearly identical to those elicited by utterances with phone durations from natural speech [43]. From this result, you may conjecture that even the simplified list of *first-order* factors above may be excessive, and that only the handful of factors implicit in the triphones themselves, supplemented by a single-phrase final-syllable coda lengthening rule, is required. This would simplify data collection and analysis for system construction.

## 15.6. PITCH GENERATION

We now describe the issues involved in generating synthetic pitch contours. Pitch, or F0, is probably the most characteristic of all the prosody dimensions. As discussed in Section 15.8, the quality of a prosody module is dominated by the quality of its pitch-generation component.

Since generating pitch contours is an incredibly complicated problem, pitch generation is often divided into two levels, with the first level computing the so-called symbolic prosody described in Section 15.4 and the second level generating pitch contours from this symbolic prosody. This division is somewhat arbitrary since, as we shall see below, a number of important prosodic phenomena do not fall cleanly on one side or the other but seem to involve aspects of both. Often it is useful to add several other attributes of the pitch contour prior to its generation, which are discussed in Section 15.6.1.

## 15.6.1. Attributes of Pitch Contours

A pitch contour is characterized not only by its symbolic prosody but also by several other attributes such as pitch range, gradient prominence, declination, and microprosody. Some of these attributes often cross into the realm of symbolic prosody. These attributes are also known in the field as phonetic prosody (termed as an analogy to phonology and phonemics).

### 15.6.1.1. Pitch Range

*Pitch range* refers to the high and low limits within which all the accent and boundary tones must be realized: a floor and ceiling, so to speak, which are typically specified in Hz. This may be considered in terms of stable, speaker-specific limits as well as in terms of an utterance or passage. For a TTS system, each voice typically has a characteristic pitch range representing some average of the pitch extremes over test utterances. This speaker-specific range can be set as an initial default for the voice or character. These limits may be changed by an application.

Another sense of pitch range is the actual exploitation of zones within the *hard* limits at any point in time for linguistic purposes, having to do with expression of the content or feeling of the message. Pitch-range variation that is correlated with emotion or other aspects of the speech event is sometimes called *paralinguistic*. This linguistic and paralinguistic use of pitch range includes aspects of both symbolic and phonetic prosody. Since it is quantitative, it certainly is a phonetic property of an utterance's F0 contour. Furthermore, it seems that most linguistic contrasts involving pitch accents, boundary tones, etc. can be realized in any pitch range. These settings can be estimated from natural speech (for research purposes) by calculating F0 mean and variance over an utterance or set of utterances, or by simply adopting the minimum and maximum measurements (perhaps the 5th and 95th percentile to minimize the effect of pitch tracker errors).

But, although pitch range is a phonetic property, it can be systematically manipulated to express states of mind and feeling in ways that other strictly phonetic properties, such as characteristic formant values, rarely are. Pitch range interacts with all the prosodic attributes you have examined above, and certain pitch-range settings may be characteristic of particular styles or utterance events. For example, it is noted [8] that: "*we cannot speak of an into-*

*nation of exclamation ... Exclamation draws impartially upon the full repertory of up-down patterns. What characterizes the class is not shape but range: exclamations reach for the extreme—usually higher but sometimes lower.*" In this sense, then, pitch range cannot be considered an arbitrary or physiological attribute—it is directly manipulated for communicative purposes.

In prosodic research, distinguishing emotive and iconic use of pitch (analogous to gesture) from strictly linguistic (logical, syntactic, and semantic expression, with arbitrary relation between signifier and signified) prosodic phenomena has been difficult. Pitch-range variation seems to straddle emotional, linguistic, and phonetic expression.

A linguistic pitch range may be narrowed or widened, and the zone of current pitch variation may be placed anywhere within a speaker's wider, physically determined range. So, for example, a male speaker might adopt a falsetto speaking style for some purpose, with his pitch range actually narrowed, but with all pitch variation realized in a high portion of his overall range, close to his physical limits.

Pitch range is a gradient property, without categorical bounds. It seems to trade off with other model components: accent, relative prominence, downstep, and declination. For example, if our model of prosody incorporates, say, an accent-strength component, but if we also recognize that pitch range can be manipulated for linguistic purposes, we may have difficulty determining, in analysis, whether a given accent is at partial strength in a wide range or at full strength in a reset, narrower range. This analytic uncertainty may be reflected in the quality of models based on the analysis.

A practical TTS system has to stay within, and make some attempt to maximize the exploitation of, the current system default or user-specified range. Thus, for general TTS purposes, the simplest approach is to use about 90% of the user-set or system default range for general prose reading, most of the time, and use the reserved 10% in special situations, such as the paragraph initial *resets*, exclamations, and emphasized words and phrases.

### 15.6.1.2. Gradient Prominence

*Gradient prominence* refers to the relative strength of a given accent position with respect to its neighbors and the current pitch-range setting. The simplest approach, where every accented syllable is realized as a H(igh) tone, at uniform strength, within an invariant range, can sound unnatural. At first glance, the prominence property of accents might appear to be a phonetic detail, in that it is quantitative, and certainly any single symbolic tonal transcription can be realized in a wide variety of relative per-accent prominence settings. However, the relative height of accents can fundamentally alter the information content of a spoken message by determining focus, contrast, and emphasis. You would hope that such linguistic content would be determined by the presence and absence, or perhaps the types (H, L, etc.), of the symbolic accents themselves. But an accented syllable at a low prominence might be perceived as unaccented in some contexts, and there is no guaranteed minimum degree of prominence for accent perception. Furthermore, as noted above, the realization of prominence of an accent is context-sensitive, depending on the current pitch-range setting.

The key knowledge deficit here is a theory of the interpretation of prominence that would allow designers to make sensible decisions. It appears that relative prominence is related to the information status of accent-bearing words and is in that sense linguistic, yet there is no theory of prominence categories that would license any abstraction. For the present, many commercial TTS systems adopt a pseudorandom pattern of alternating stronger/weaker prominence, simply to avoid monotony. If a word is tagged for emphasis, or if its information status can otherwise be inferred, its prominence can be heightened within the local range.

In the absence of information on the relative semantic salience of accented words in the utterance, successive prominence levels are varied in some simple alternating pattern, to avoid monotony. Rather than limiting the system to a single peak F0 value per accented syllable, several points could be specified, which, when connected by interpolation and smoothing, could give varied effects within the syllable, such as rising, falling, and scooped accents.

### 15.6.1.3. Declination

Related to both pitch range and gradient prominence is the long-term downward trend of accent heights across a typical reading-style, semantically neutral, declarative sentence. This is called *declination*. Although this tendency, if overdone, can simply give the effect of a bored or uncomprehending reader, it is a favorite prosodic effect for TTS systems, because it is simple to implement and licenses some pitch change across a single sentence. If a system uses a 'top line' as a reference for calculating the height of every accent, the slope of that top line can simply be declined across the utterance. Otherwise, each accent's prominence can be realized as a certain percentage of the height of the preceding one. Declination can be reset at utterance boundaries, or within an utterance at the boundaries of certain linguistic structures, such as the beginning of quoted speech. Intrasentence phrase and clause types that typically narrow the pitch range, such as parentheticals and certain relative clauses, can be modeled by suspending the declination, or adopting a new declination line for the temporary narrowed range, then resuming the suspended longer-term trend as the utterance progresses. Needless to say, declination is not a prominent feature of spontaneous speech and in any case should not be overdone.

The minor effect of declination should not be confused with the tendency in all kinds of nonquestioning utterances to end with a very low pitch, close to the bottom of the speaker's range. In prosodic research this is called *final lowering* and is well-attested as a phenomenon that is independent of declination [29]. The ToBI notation used to specify final lowering is the complex boundary tone L-L%. In Figure 15.6 we show the declination line together with the other two *downers of intonation*: downstep and final lowering described in Section 15.4.4.

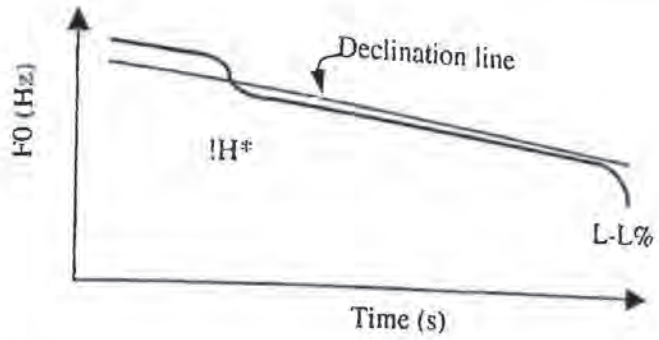



Figure 15.6 The three downers of intonation: the declination line, a downstep (!H*), and the final lowering (L-L%).

### 15.6.1.4. Phonetic F0—Microprosody

*Microprosody* refers to those aspects of the pitch contour that are unambiguously phonetic and that often involve some interaction with the speech carrier phones. These may be regarded as *second-order* effects, in the sense that rendering them well cannot compensate for incorrect accentuation or other mistakes at the symbolic level. Conversely, making no attempt to model these but putting a great deal of care into the semantic basis for determining accentuation, contrast, focus, emphasis, phrasing, etc. can result in a system of reasonable quality. Nevertheless, all else being equal, it is advisable to make some attempt to capture the local phonetic properties of natural pitch contours.

If the strength of accents is controlled semantically, by having equal degrees of focus on words of differing phonetic makeup, it has been observed that high vowels described in Chapter 2 carrying H* accents are uniformly higher in the phonetic pitch range than low vowels with the same kinds of accent. The distinction between high and low vowels correlates with the position of the tongue in articulation (high or low in the mouth). The highest English vowels by this metric are */iy/* (as in *bee*) and */uw/* (as in *too*), while the lowest vowel is */aa/* as in *father*. The predictability of F0 under these conditions may relate to the degree of tension placed on the laryngeal mechanisms by the raised tongue position in the high vowels as opposed to the low. In any case, this effect, while probably perceptually important in natural speech, is challenging for a synthesizer. The reason relates again to the issue of gradient prominence, discussed above. Apart from experimental prompts in the lab, there is currently no principled way to assign prominence for accent height realization based on utterance content in general TTS. It may therefore be difficult for a listener to correctly factor pitch accent height that is due to correctly (or incorrectly) assigned gradient prominence from height variation related to the lower-level phonetic effects of vowel height.

Another phonetic effect is the level F0 in the early portion of a vowel that follows a voiced obstruent such as /b/, contrasted with the typical fall in F0 following a voiceless obstruent such as /p/. This phonetic conditioning effect, of both preceding and following consonants, can be observed most clearly when identical underlying accent types are assigned to the carrier vowel, and may persist as long as 50 ms or more into the vowel. The exact contribution of the pre-vocalic consonant, the post-vocalic consonant, and the underlying accent type are difficult to untangle, though [54] is a good survey of all research in this area and adds new experimental results. For commercial synthesizers, this is definitely a second-order effect and is probably more important for rule-based formant synthesizers (see Chapter 16), which need to use every possible cue to enforce distinctions among consonants in phoneme perception, than for strictly intonational synthesis. However, in order to achieve completely natural prosody in the future, this area will have to be addressed.

Last, and perhaps least, *jitter* is a variation of individual cycle lengths in pitch-period measurement, and *shimmer* is variation in energy values of the individual cycles. These are distinct concepts, though somewhat correlated. Obviously, this is an influence of glottal pulse shape and strength on the quality of vowels. Speech with jitter and shimmer over 15% sounds pathological, but complete regularity in the glottal pulse may sound unnatural. For a deeper understanding of how these could be controlled, see Chapter 16.

### 15.6.2. Baseline F0 Contour Generation

We now examine a simple system that generates F0 contours. Although each stage of an F0 contour algorithm ideally requires a complete natural language and semantic analysis system, in practice a number of rules are often used. The system described here illustrates most of the important features common to the pitch-generation systems of commercial synthesizers.

First, let's consider a natural speech sample and describe what initial information is needed to characterize it, and how an artificial pitch contour can be synthesized based on the input analysis. The chosen sample is the utterance *"Don't hit it to Joey!"*, an exclamation, from the ToBI Labeling Guidelines sample utterance set [4]. The natural waveform, aligned pitch contour, and abstract ToBI labels are shown in Figure 15.7. This utterance is about 1.63 seconds and it has three major ToBI pitch events:

| | |
|---|---|
| H* | high pitch accent on *Don't* |
| L*+!H | low pitch accent with following downstepped high on *Joey* |
| L-L% | low utterance-final boundary tone at the very end of utterance |

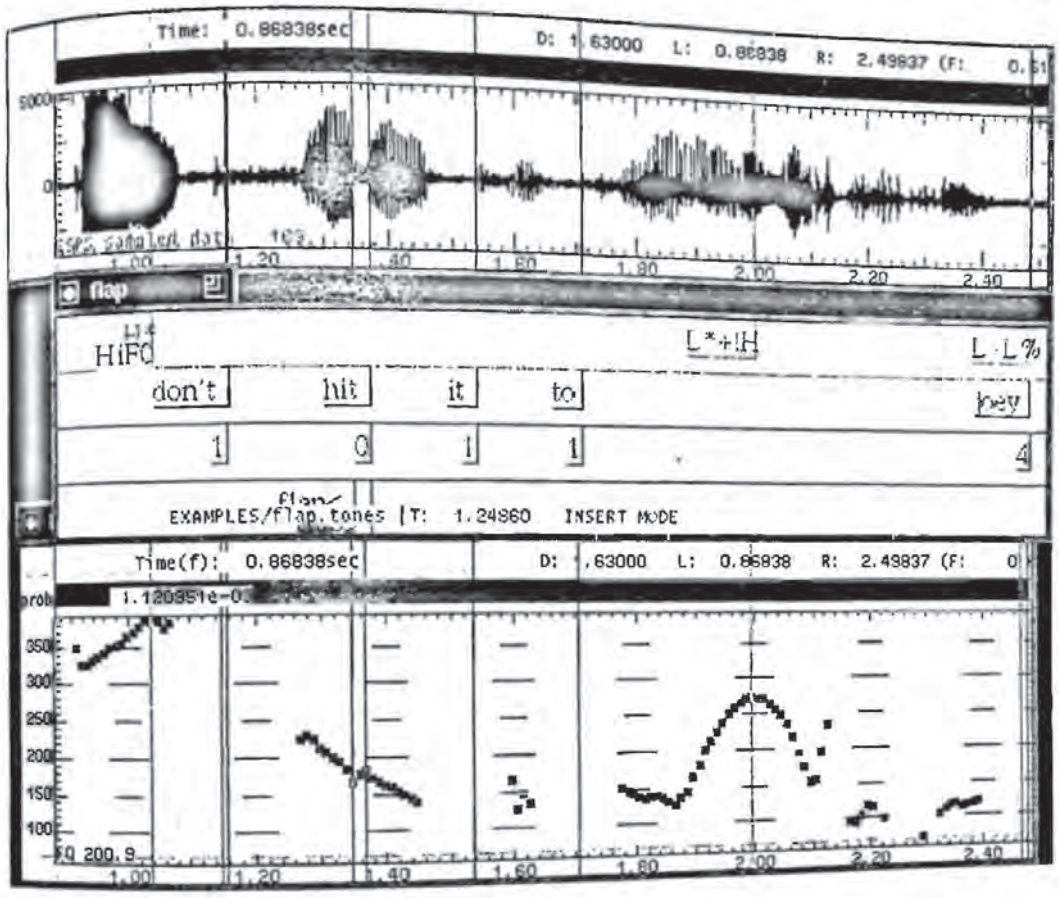



**Figure 15.7** Time waveform, segmentation, ToBI marks, and pitch contour for the utterance *"Don't hit it to Joey!"* spoken by a female speaker (after Beckman [4]).

The input to the F0 contour generator includes:

- Word segmentation.
- Phone labels within words.
- Durations for phones, in milliseconds.
- Utterance type and/or punctuation information.
- Relative salience of words as determined by grammatical/semantic analysis.
- Current pitch-range settings for voice.

### 15.6.2.1. Accent Determination

Although accent determination ideally requires a complete natural language and semantic analysis system (see Section 15.4.3), in practice a number of rules are often used. The first rule is: *Content word categories of noun, verb, adjective, and adverb are to be accented, while the function word categories (everything else, such as pronoun, preposition, conjunction, etc.) are to be left unaccented.* Rules can be used to tune this by specifying which POS is accented or not and in which context.

If we apply that simple metric to the natural sample of Figure 15.7, we see that it does not account for the accentuation of 'hit', which, as a verb, should have been accented. In a real system perhaps we would have accented it, and this might have resulted in the typical overaccented quality of synthetic prosody. For this sample discussion, let's adopt a simplified version of a rule found in some commercial synthesizers: *Monosyllabic common verbs are left unaccented.*

What about *"Don't"*? A simplistic view would state that the POS-based policy has done the right thing, after all *"Don't"* can be regarded as a verbal form. However, usually *do* is considered an auxiliary verb and is not accented. For now we adopt another rule that says: *In a negative imperative exclamation, determined by presence of a second-person negative auxiliary form and a terminal exclamation point, the negative term gets accented.* The adoption of these corollaries to the simple POS-based accentuation rule accounts for our accent placement in the present example, but of course it sometimes fails, as does any rigid policy. So our utterance would now appear (with words selected for accent in upper case) as *"DON'T hit it to JOEY!"*

### 15.6.2.2. Tone Determination

In the limit, tone determination (see Section 15.4.4) also requires a complete natural language and semantic analysis system, but in practice a number of rules are often used. Generally, in working systems, H* is used for all pitch accent tones, and this is actually very realistic, as H* is the most frequent tone in natural speech.

Sometimes complex tones of the type L*+!H are thrown in for a kind of pseudovariety in TTS. In our sample natural utterance this is the tone that is used, so here we assume that this is the accent type assigned.

We also need to mark punctuation-adjacent and utterance-final phonemes as rise, continue, or fall boundaries. In this case we mark it as L-L%.

### 15.6.2.3. Pitch Range

To determine the pitch range, we are going to make use of three lines as a frame within which all pitches are calculated. The top line and bottom line would presumably be derived from the current or default pitch-range settings as controlled by an application or user. Here

we set them in accord with the limits of our natural sample. Note that while, for this example, the pitch contour is generated within an actual pitch range, it could also be done within an abstract range of, say, 1–100, which the voice-generation module could map to the current actual setting. So we set the top line at $T = 375$ Hz and the base line at $B = 100$ Hz.

It is more advantageous to work in a logarithmic scale, because it is more easily ported from males to females, and because this better represents human prosody. There are 24 semitones in an octave; thus a semitone corresponds to a ratio of $a = 2^{1/24}$. The pitch range can be expressed in semitones as

$$n = 24 \log_2 (T/B) = 80 \log_{10} (T/B) \tag{15.4}$$

so that we can express frequencies in semitones as

$$f_0 = 80 \log_{10} F_0 \tag{15.5}$$

and its inverse

$$F_0 = 10^{f_0/80} \tag{15.6}$$

Using Eq. (15.5), the top line is $t = 206$ and the base line is $b = 160$. The reference line is a kind of midline for the range, used in the accent scaling calculations, and is set halfway between the bottom and top lines, i.e., $r = 183$, and using Eq. (15.6), $R = 194$ Hz.

### 15.6.2.4. Prominence Determination

The relative prominence of the words (see Section 15.6.1.2) allows the pitch module to scale the pitch within any given pitch range. Here we assume (arbitrarily) that $N = 5$ degrees of abstract relative prominence are sufficient. This means that, e.g., an H* pitch accent with prominence 5 will be at or near the very top of the current pitch-range setting, while an L* with the same prominence will be at or near the very bottom of the range. Smaller prominence numbers indicate less salience, placing their pitch events closer to the middle of the range.

Converting the abstract tone types plus prominence into pitch numbers is more art than science (but see Section 15.6.4 for a discussion of data-based methods for this process). Here we assume a simple linear relationship between the tone's type and relative prominence:

$$f_0[i] = r + (t - r) * p[i]/N \tag{15.7}$$

In the limit, prominence determination also requires a complete natural language and semantic analysis system, but in practice a number of heuristics are often used. One such heuristic is: *In a negative imperative exclamation, the negative term gets the most emphasis,* leading to a relative prominence assignment of 5 on 'don't.' Using Eqs. (15.7) and (15.6), the anchor equals the top range of 375 Hz.

Then, since the L*+!H involves a *downstepped* term, it must by definition be lower than the preceding H* accent, so we arbitrarily assign it a relative prominence of '2'. The L*+!H is more complex, requiring calculation and placement of two separate anchor points. For simplicity we are using a single prominence value for complex tones like L*+!H, but we could also use a value-per-tone approach, at the cost of greater analytical complexity. Using Eq. (15.7), it corresponds to 192 semitones, and with Eq. (15.6), the value of !H is 251 Hz. For L*, we use a prominence of –2 (we use negative values for L tones), which, using Eq. (15.7), results in an anchor of 174, or alternatively 149 Hz.

The L-L% tone is a boundary tone, so it always goes on the final frame of a syllable-final (in this case, utterance-final) phone. The L-L% in most ToBI systems is not treated as a two-part complex tone but rather as a *super low* L% boundary, falling at the very bottom of the speaker's pitch range, i.e., prominence of 5, for a few frames. Thus the F0 value of these anchor point is 100 Hz.

We also need to set anchors for the initial point. The initial anchor is usually set at some arbitrary but high place within the speaker's range (perhaps a rule looking at u tterance type can be used). A prominence of 4 can be used, yielding a value of 329 Hz.

Finally we need to determine where to place the anchors within the accented syllable. Often they are placed in the middle of the vowel. All the anchor points are shown in Figure 15.8.

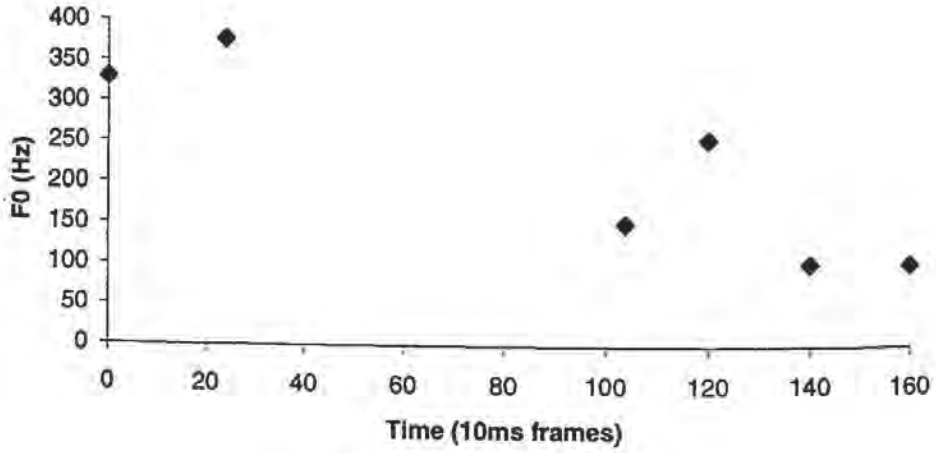



**Figure 15.8** Anchor points of the F0 contour.

### 15.6.2.5. F0 Contour Interpolation

To obtain the full F0 contour we need some kind of interpolation. One way is to interpolate linearly and follow with a multipoint moving-average window over the resulting (angular) contour to smooth it out. Another possibility is a higher-order interpolation polynomial. In this case a cubic interpolation routine is called, which has the advantage of retaining the exact anchor points in a smoothed final contour (as opposed to moving average, which smears the anchor points). In general the choice of interpolation algorithm makes little per-

ceptual difference, as long as no sharp 'corners' remain in the contour. In Figure 15.9 the contour was interpolated fully, without regard to voicing properties of underlying phones. In the graph, the sections corresponding to unvoiced phones have been replaced with zero, for ease of comparison to the sample in Figure 15.7. The interpolation can be done in the linear frequency, as in Figure 15.9, or in the log-frequency.

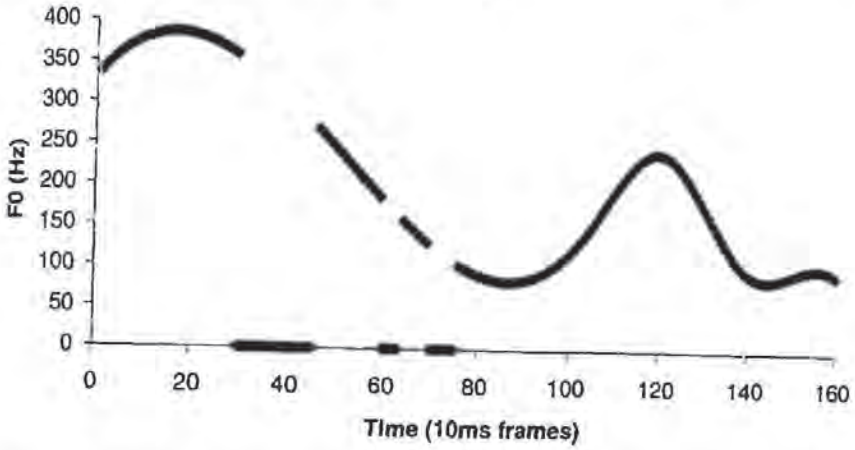



**Figure 15.9** F0 contour of Figure 15.8 after cubic interpolation. Sections corresponding to unvoiced phones have been replaced with zero.

In order for the interpolation algorithm to operate properly we need to have phone durations so that the anchor points are appropriately spaced apart. In this baseline algorithm, we followed the algorithm described in Section 15.5.2.

### 15.6.2.6. Interface to Synthesis Module

Finally, most synthesizers cannot accept an arbitrary number of pitch controls on a given phoneme, nor it this necessary. We can downsample the pitch buffer to allocate a few characteristic points per phoneme record, and, if the synthesizer can interpolate pitch, it may be desirable to skip pitch controls for unvoiced phones altogether. The F0 targets can be placed at default locations (such as the left edge and middle of each phone), or the placements can be indicated by percent values on each target, depending on what the synthesizer supports. This has to be in agreement with the specific interface between the prosody module and the synthesis module as described in Section 15.2.

### 15.6.2.7. Evaluation and Possible Improvements

In comparing the output contour of Figure 15.9 to the natural one of Figure 15.7, how well have we done? As a first-order approximation, from visual inspection, it is somewhat similar

to the original. Of course, we have used hand-coded information for the accent property, accent type, and prominence! However, these choices were reasonable, and could apply them as defaults to many other utterances. At a minimum, almost exactly the code given above would apply without change and give a decent contour for a whole 'family' of similar utterances, such as *"Don't hit the ball to Joey!"* or *"Never give the baseball to Henry!"* A higher-order discourse module would need to determine that *ball* and *baseball* are not accented, however, in order to use the given contour with the same rhetorical effect (presumably *ball* and *baseball* in these cases could be given/understood information).

Something very much like the system described here has been used in most commercially marketed synthesizers throughout the 1990s. This model seems overly simple, even crude, and presumably it could be substantially augmented, or completely replaced by something more sophisticated.

However, many weaknesses are apparent also. For one thing, the contour appears very smooth. The slight *jitter* of real contours can be easily simulated at a final stage of pitch buffer processing by modifying +/- 3 or 4 Hz to the final value of each frame. The degree to which such niceties actually affect listener perceptions depend entirely on the quality of the synthetic speech and the quality of the pitch-modification algorithms in the synthesizer.

The details of peak placement obviously differ between the natural and synthetic contours. This is partly due to the crude uniform durations used, but in practice synthesizers may incorporate large batteries of rules to decide exactly (for example) which frame of a phone the H* definition point should appear in—early, middle, late? Sometimes this decision is based on surrounding phonetic structure, word and syllable structure, and prosodic context. The degree to which this matters in perception depends partly on synthetic speech quality overall.

### 15.6.3. Parametric F0 Generation

To realize all the prosodic effects discussed above, some systems make almost direct use of a real speaker's measured data, via table lookup methods. Other systems use data indirectly, via parametrized algorithms with generic structure. The simplest systems use an invariant algorithm that has no particular connection to any single speaker's data, such as the algorithm described in the baseline F0 generation system of Section 15.6.2. Each of these approaches has advantages and disadvantages, and none of them has resulted in a system that fully mimics human prosodic performance to the satisfaction of all listeners. As in other areas of TTS, researchers have not converged on any single standard family of approaches. Once we venture beyond the simplest approaches, we find an interesting variety of systems, based on different assumptions, with differing characteristics. We now discuss a few of the more representative approaches.

Even models that make little or no attempt to analyze the internal components of an F0 contour must be indexed somehow. System designers should choose indexing or predictive factors that are derivable from text analysis, are general enough to cover most prosodic situations, and are powerful enough to specify high-quality prosody. In practice, most mod-

els' predictive factors have a rough correspondence to, or are an elaboration of, the elements of the baseline algorithm of Section 15.6.2. A typical list might include the following:

- Word structure (stress, phones, syllabification)
- Word class and/or POS
- Punctuation and prosodic phrasing
- Local syntactic structure
- Clause and sentence *type* (declarative, question, exclamation, quote, etc.)
- Externally specified focus and emphasis
- Externally specified speech style, pragmatic style, emotional tone, and speech act goals

These factors jointly determine an output contour's characteristics, as listed below. Ideally, any or all of these may be externally and directly specified, or they may be inferred or implied within the F0 generation model itself:

- Pitch-range setting
- Gradient, relative prominence on each syllable
- Global declination trend, if any
- Local shape of F0 movement
- Timing of F0 events relative to phone (carrier) structure

The combinatorial complexity of these predictive factors, and the size of the resulting models, can be serious issues for practical systems that strive for high coverage of prosodic variability and high-quality output. The possibility of using *externally specified* symbolic markups gives the whole system a degree of modularity, in that prosodic annotation can be specified directly by an authoritative outside source or can be derived automatically by the symbolic prosody prediction process that precedes F0 contour generation.

Parametric models propose an underlying architecture of prosodic production or perception that constrains the set of possible outputs to conform to universal constants of the human speech mechanism. Naturally, these models need settings to distinguish different speakers, different styles, and the specifics of utterances. We describe superposition models and ToBI Realization models.

### 15.6.3.1. Superposition Models

An influential class of parametric models was initiated by the work [35] for Swedish, which proposed additive superposition of component contours to synthesize a complex final F0 track. In the version refined and elaborated in [14], the component contours, which may all have different strengths and decay characteristics, may correspond to longer-term trends, such as phrase or utterance declination, as well as shorter-time events, such as pitch accents

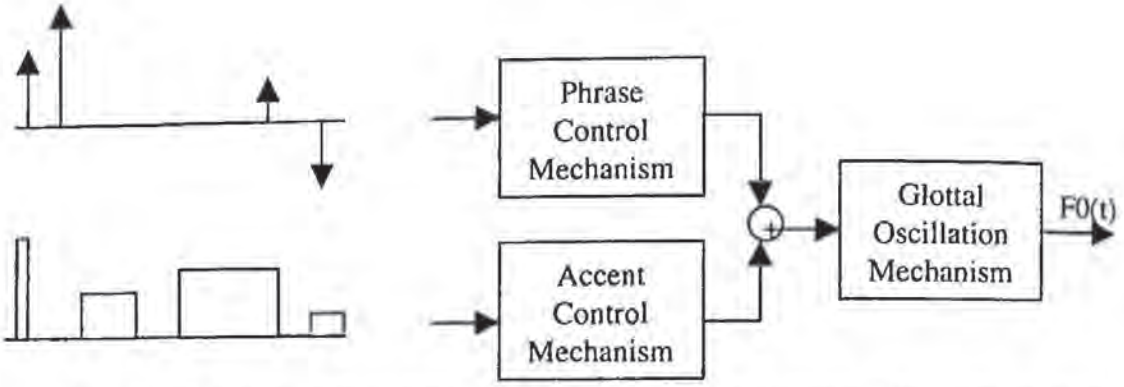



**Figure 15.10** Fujisaki pitch model [15]. F0 is a superposition of phrase effects with accent effects. The phrase mechanism controls things like the declination of a declarative sentence or a question, whereas the accent mechanism accounts for accents in individual syllables.

on words. The component contours are modeled as the critically damped responses of second-order systems to impulse functions for the longer-term, slowly decaying phrasal trends, and step or rectangular functions of shorter-term accent events. The components so generated are added and ride a baseline that is speaker specific. The basic ingredients of the system, known as Fujisaki's model [15, 19], are shown in Figure 15.10. The resulting contour is shown in Figure 15.11. Obviously, similar effects can be generated with linear accent shapes as described in the simpler model above, with smoothing. However, there are some plausible claims for the articulatory correlates of the constraints imposed in the second-order damping and superposition effects of this model [33].

Superposition models of this type can, if supplied with accurate parameters in the form of time alignments and strengths of the impulses and steps, generate contours closely mimicking natural examples. In this respect, the remaining quality gap for general application is in the parametric knowledge driving the model, not in the model structure per se. These kinds of models have been particularly successful in replicating the relatively constrained Japanese reading-style. Whether these models can account straightforwardly for the immense variety of a large range of English speakers and text genre, or whether, on the contrary, the parameters proliferate and the settings become increasingly arbitrary, remains to be seen.



**Figure 15.11** Composite contour obtained by low-pass filtering the impulses and boxes in the Fujisaki model of Figure 15.10.

### 15.6.3.2. ToBI Realization Models

One simple parametric model, which in its inherent structure makes only modest claims for principled correspondence to perceptual or articulatory reality, is designed to support prosodic symbols such as the *Tones and Break Indices* (ToBI) system. This model, variants of which are developed in [2, 54], posits two or three control lines, by reference to which ToBI-like prosody symbols can be scaled. This provides for some independence between symbolic and phonetic prosodic subsystems. In the model shown in Figure 15.12, the top line is an upper limit of the pitch range. It can be slanted down to simulate declination. The bottom line represents the bottom of the speaker's range. Pitch accents and boundary tones (as in ToBI) are scaled from a reference line, which is often midway in the range in a logarithmic scale of the pitch range, as described in the baseline algorithm of 15.6.2. You can think of this scaling as operating within a percentage of the current range, rather than absolute values, so a generic method can be applied to any arbitrary pitch-range setting. The quantitative instantiation of accent height is done at the final stage. The accents and boundary tones consist of one or more points, which can be aligned with the carrier phones; then interpolation is applied between points, and smoothing is performed over the resulting contour.

In Figure 15.12, $t$, $r$, and $b$ are the top, reference, and baseline pitch values, respectively. They are set from the defaults of the voice character and by user choice. The base $b$ is considered a physiological constraint of voice. $P$ is the prominence of the accent and $N$ is the number of prominence steps. Declination can be modeled by slanting the top and/or reference lines down. The lowered position of the reference in Figure 15.12 reflects the observation that the realization of H(igh) and L(ow) ToBI abstract tones in a given pitch range is asymmetric, with a greater portion available for H, while L saturates more quickly. This is why placing the reference line midway between the top and base lines in a log-frequency scale automatically takes care of this phenomenon. After target points are located and scaled according to their gradient prominence specifications, the (hopefully sparse) targets can be interpolated and the resulting contour smoothed. If the contour is calculated in, say, 10-ms frames, two pitch targets sampled from the contour vector per phone usually suffice to reproduce the intended prosodic effects faithfully.

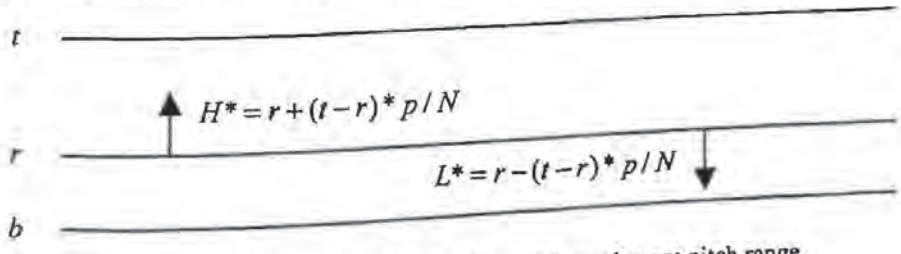



**Figure 15.12** A typical model of tone scaling with an abstract pitch range.

If a database of recorded utterances with phone labeling and F0 measurements has been reliably labeled with ToBI pitch annotations, it may be possible to automate the implementation of the ToBI-style parametrized model. This was attempted with some success in [5], where linear regression was used to predict syllable initial, vowel medial, and syllable final F0 based on simple, accurately measurable factors such as:

- ToBI accent type of target and neighbor syllables
- ToBI boundary pitch type of target and neighbor syllables
- Break index on target and neighbor syllables
- Lexical stress of target and neighbor syllables
- Number of syllables in phrase
- Target syllable position phrase
- Number and location of stressed syllable(s)
- Number and location of accented syllable(s)

Models of this sort do not incorporate an explicit mechanism (like the scaling direction from $r$ in Figure 15.12) to distinguish H(igh) from L(ow) tone space, beyond what the data and its annotations imply.

The work in [47] consists of a ToBI realization model in which the 'smoothing' mechanism is built-in as a dynamical system whose parameters are also learnt from data. This work could be viewed as a stochastic realization of Fujisaki's superposition model without the phrase controls and where the accents are given by ToBI labels.

Both the ToBI realization models and the superposition models could, if supplied with sufficiently accurate measurements of an example contour, reproduce it fairly accurately. Both models require much detailed knowledge (global and local pitch range; location, type, and relative strength of accents and boundary tones; degree of declination; etc.) to function at human-level quality for a given utterance. If a system designer is in possession of a completely annotated, high-quality database of fully representative prosodic forms for his/her needs, the question of deployment of the database in a model can be made based on performance tradeoffs, maintenance issues, and other engineering considerations. If, on the other hand, no such database is available for the given application purpose, extremely high prosodic quality, including lively yet principled variation, should not be expected to result simply from choosing the 'mathematically correct' model type.

### 15.6.4. Corpus-Based F0 Generation

It is possible to have F0 parameters trained from a corpus of natural recordings. The simplest models are the direct models, where an exact match is required. Models that offer more generalization have a library of F0 contours that are indexed either from features from the parse tree or from ToBI labels. Finally, there are F0 generation models from a statistical

network such as a neural network or an HMM. In all cases, once the model is set, the parameters are learned automatically from data.

#### 15.6.4.1. Transplanted Prosody

The most direct approach of all is to store a single contour from a real speaker's utterance corresponding to every possible input utterance that one's TTS system will ever face. This seems to limit the ability to freely synthesize any input text. However, this approach can be viable under certain special conditions and limitations. These controls are so detailed that they are tedious to write manually. Fortunately, they can be generated automatically by speech recognition algorithms.

When these controls (*transplanted prosody*), taken from an authentic digitized utterance, are applied to synthetic voice units, the results can be very convincing, sometimes nearly as good as the original digitized samples [43]. A system with this capability can mix predefined utterances having natural-quality prosody, such as greetings, with flexible synthesis capabilities for system response, using a consistent synthetic voice. The transplanted prosody for the *frozen* phrases can be derived either from the original voice data donor used to create the synthetic voice model, or any other speaker, with global adjustment for pitch-range differences. Another use of the transplanted prosody capability is to compress a spoken message into ASCII (phone labels plus the prosodic controls) for playback, preserving much of the quality, if not the full individuality, of the original speaker's recording.

#### 15.6.4.2. F0 Contours Indexed by Parsed Text

In a more generalized variant of the direct approach, once could imagine collecting and indexing a gigantic database of clauses, phrases, words, or syllables, and then annotating all units with their salient prosodic features. If the terms of annotation (word structure, POS, syntactic context, etc.) can be applied to new utterances at runtime, a prosodic description for the closest matching database unit can be recovered and applied to the input utterance [23]. The advantages here are that prosodic quality can be made arbitrarily high, by collecting enough exemplars to cover arbitrarily large quantities of input text, and that detailed analysis of the deeper properties of the prosodic phenomena can be sidestepped. The potential disadvantages are:

- Data-collection time is long (which affects the capability to create new voices).
- A large amount of runtime storage is required (presumably less important as technology progresses).
- Database annotation may have to be manual, or if automated, may be of poor quality.
- The model cannot be easily modified/extended, owing to lack of fundamental understanding.

- Coverage can never be complete, therefore rulelike generalization, fuzzy match capability, or back-off, is needed.
- Consistency control for the prosodic attributes (to prevent unit boundary mismatches) can be difficult.

The first two disadvantages are self-explanatory. The difficulty of annotating the database, to form the basis of the indexing and retrieval scheme, depends on the type and depth of the indexing parameters chosen. Any such scheme requires annotations to identify the component phones of each unit (syllable, word, or phrase) and their durations. This can usually be obtained from speech recognition tools [23], which may be independently required to create a synthetic voice (see Chapter 16). Lexical or word stress attributes can be extracted from an online dictionary or NLP system, though, as we have seen above, lexical stress is neither a necessary nor a sufficient condition for predicting pitch accent placement.

If only a very high level of description is sought, based primarily on the pragmatics of utterance use and some syntactic typology, it may not be necessary to recover a detailed symbolic pitch analysis. An input text can be described in high-level pragmatic/semantic terms, and pitch from the nearest matching word or phrase from the database can be applied with the expectation that its contour is likely correct. For example, such a system might have multiple prosodic versions of a word that can be used in different pragmatic senses, such as *ok*, which could be a question, a statement, an exclamation, a signal of hesitation or uncertainty, etc. The correct version must be selected based on the runtime requirements of the application.

Direct prosody schemes of this type often preserve the original phone carrier material of each instance in order to assure optimal match between prosody and spectrum. However, with DSP techniques enabling arbitrary modifications of waveforms (see Chapter 16), this is not strictly necessary; the prosodic annotations could stand alone, with phone label annotation only. If more detailed prosodic control is required, such as being aware of the type of accent, its pitch range, prominence, and other features, the annotation task is much more difficult.

A straightforward and elegant formulation of the lookup-table direct model approach can be found in [30]. This system, created for Spanish but generally adaptable, is based on a large single-speaker recorded database of a variety of sentence types. The sentences are linguistically analyzed, and prosodic structure is hypothesized based on syllables, accent groups (groups of syllables with one lexical stress), breath groups (groups of accent-groups between pauses regardless of the duration of the pause), and sentences. Note that these structures are hypothesized based on the textual material alone, and the speaker will not always perform accordingly. Pitch (initial, mid, and final F0) and duration data for each spoken syllable is automatically measured and stored. At runtime, the input sentence is analyzed using the same set of structural attributes inferred from the text, and a vector of candidate syllables from the database is constructed with identical, or similar, structural context and attributes for each successive input syllable position.

The best path through the set of such candidates is selected by minimizing the F0 distance and disjuncture across the utterance. This is a clean and simple approach to jointly

utilizing both shallow linguistic features and genuine phonetic data (duration and F0), with dynamic programming to smooth and integrate the output F0 contour. However, as with any direct modeling approach, it lacks significant generalization capabilities outside the textual material and speaking style specified during the data collection phase, so a number of separate models may have to be constructed.

The CHATR system of ATR (Japan) [9] takes a similar approach, in that optimal prosody is selected from prerecorded units, rather than synthesized from a more general model. The CHATR system consists of a large database of digitized speech, indexed by the speaker identity, the phoneme sequences of the words, and some pragmatic and semantic attributes. Selection of phonemes proceeds by choosing the minimal-cost path from among the similarly indexed database candidate units available for each phoneme or longer segment of speech to be synthesized. This system achieves high quality by allowing the carrier phones to bear only their original prosody—pitch modification of the contour is minimized or eliminated. Of course, the restriction of DSP modification implies a limitation of the generalizability of the database. This type of approach obtains the prosody implicitly from the database [31], and as such combines both the prosody and speech synthesis modules. This type of minimal-cost search is described in more detail in Chapter 16.

### 15.6.4.3. F0 Contours Indexed by ToBI

The architecture for a simple and straightforward direct model indexed by ToBI is diagrammed in Figure 15.13. This model combines the two often-conflicting goals: it is empirically (corpus) based, but it permits specification in terms of principled abstract prosodic categories. In this model, an utterance to be synthesized is annotated in terms of its linguistic features—perhaps POS, syntactic structure, word emphasis (based on information structure), etc. The utterance so characterized is matched against a corpus of actual utterances that are annotated with linguistic features and ToBI symbols, Corpus (a). A *fuzzy* matching capability based on edit distance or dynamic programming can be incorporated. If Corpus (a) is sufficiently large and varied, a number of possible ToBI renderings of either the entire utterance or selected parts of it may be recovered. At this level of abstraction, the ToBI labels would not encode relative prominence specifications (strength of pitch extrusions) or pitch range. The set of such abstractly described contours can then be fuzzy matched into Corpus (b), a set of ToBI annotated actual contours, and the best set of matches recovered.

Note that while it is possible that Corpus (a) is the exact same base material as Corpus (b), the model does not enforce an identity, and there may be reasons to desire such flexibility and modularity, depending on the degree and quality of data and annotation at each level. Once a number of likely actual contours have been identified, they can be passed to a voice-unit selection module. The module can select the combination of segmental strings (sometimes called 'long units,' since they may combine more than one phoneme) from the voice database whose original prosody is closest to one of the candidate contours, using root-mean-square-error, correlation, or other statistical tests. Those units are then concatenated (with their prosody unmodified) and sent to the application or played out.

A model of this type has some of the disadvantages of direct models as listed above. It also assumes availability of large and varied databases of both prosodic contours and segmental (phone) long units for concatenation (see Chapter 16). It further requires that these databases be annotated, either by human labelers or automated systems. However, it has certain advantages as well:

- It allows for symbolic, phonological coding of prosody.
- It has high-quality natural contours.
- It has high-quality phonetic units, with unmodified pitch.
- Its modular architecture can work with user-supplied prosodic symbols.

It also allows the immediate, temporary use of data that is collected for deeper analysis, in the hope of eventual construction of smaller, parametrized models. The model of Figure 15.13 is a generalization of the prosody system described in [23].

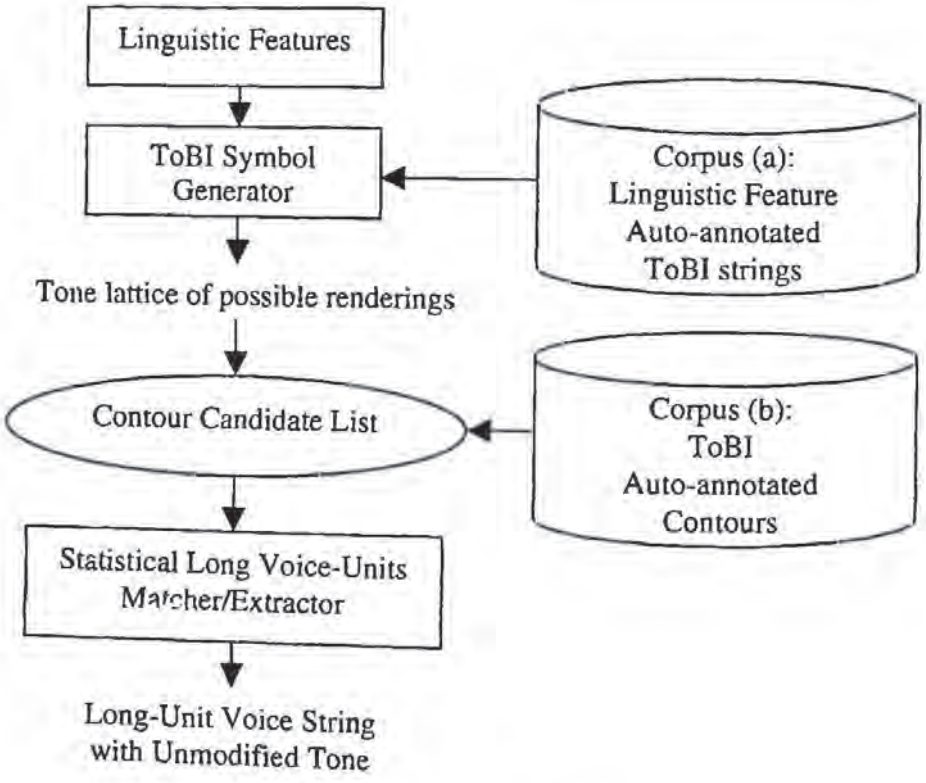



**Figure 15.13** A corpus-based prosodic generation model.

## 15.7. PROSODY MARKUP LANGUAGES

Chapter 14 discussed generalized document markup schemes for text analysis. Most TTS engines provide simple text tags and application programming interface controls that allow at least rudimentary hints to be passed along from an application to a TTS engine. We expect to see more sophisticated speech-specific annotation systems, which eventually incorporate current research on the use of semantically structured inputs to synthesizers, sometimes called concept-to-speech systems. A standard set of prosodic annotation tags would likely include tags for insertion of silence pause, emotion, pitch baseline and range, speed in words-per-minute, and volume. This would be in addition to general tags for specifying the language of origin if not predictable, character of the voice, and text normalization context such as address, date, email, etc.

For prosodic processing, text may be marked with tags that have scope, in the general fashion of XML. Some examples of the form and function of a few common TTS tags for prosodic processing, based loosely on the proposals of [65], are introduced below. Other tags can be added by intermediate subcomponents to indicate variables such as accents and tones. This extension allows for even finer research and prosody models.

- **Pause** or **Break** commands might accept either an absolute duration of silence in milliseconds, or, as in the W3C proposal, a mnemonic describing the relative salience of the pause (Large, Medium, Small, None), or a *prosodic punctuation* symbol from the set ',', '.', '?', '!', '...', etc., which not only indicates a pause insertion but also influences the typical pitch contour of the phone segments entering and leaving the pause area. For example, specifying ',' as the argument of a Pause command might determine the use of a continuation rise on the phones immediately preceding the pause, indicating incompletion or listing intonation.
- **Rate** controls the speed of output. The usual measurement is *words per minute*, which can be a bit vague, since words are of very different durations. However, this metric is familiar to many TTS users and works reasonably well in practice. For non-IndoEuropean languages, different metrics must be contemplated. Some power listeners who use a TTS system routinely can tolerate (in fact, demand) rates of over 300 words per minute, while 150 or fewer might be all that a novice listener could expect to reliably comprehend.
- **Baseline Pitch** specifies the desired average pitch: a level around which, or up from which, pitch is to fluctuate.
- **Pitch Range** specifies within what bounds around the baseline pitch level line the pitch is to fluctuate.
- **Pitch** commands can override the system's default prosody, giving an application or document author greater control. Generally, TTS engines require some freedom to express their typical pitch patterns within the broad limits specified by a Pitch markup.

- **Emphasis** emphasizes or deemphasizes one or more words, signaling their relative importance in an utterance. Its scope could be indicated by XML style. Control over emphasis brings up a number of interesting considerations. For one thing, it may be desirable to have degrees of emphasis. The notion of gradient prominence—the apparent fact that there are no categorical constraints on levels of relative emphasis or accentuation—has been a perpetual thorn in the side for prosodic researchers. This means that in principle any positive real number could be used as an argument to this tag. In practice, most TTS engines would artificially constrain the range of emphasis to a smaller set of integers, or perhaps use semantic labels, such as *strong*, *moderate*, *weak*, *none* for degree of emphasis. Emphasis may be realized with multiple phonetic cues. Thus, if the user or application has, for example, set the pitch range very narrowly, the emphasis effect may be achieved by manipulation of segmental duration or even relative amplitude. The implementation of emphasis by a TTS engine for a given word may involve manipulation (e.g., de-accentuation) of surrounding words as much as it involves heightening the pitch or volume, or stretching the phone durations, of the target word itself. In most cases the main phonetic and perceptual effect of emphasis or accentuation is heard on the lexically main stressed syllable of the word, but this can be violated under special conditions of semantic focus, e.g., "*I didn't say employer, I said employee.*" This would require a more powerful emphasis specification than is currently provided in most TTS systems, but alternatively it could be specified using phone input commands such as "*The <emp>truth</emp>, the <emp>whole truth</emp>, and nothing <emp> but</emp> the truth.*" For more control, future TTS systems may support degree emphasis: "*... nothing <emp level="strong">but</emp> the truth*" or even deemphasis: "*... nothing <emp level= "reduced">but</emp> the truth*". Emphasis is related to prominence, discussed in Section 15.6.1.2.

## 15.8. PROSODY EVALUATION

Evaluation of a complete TTS system is discussed in Chapter 16. We limit ourselves here to evaluating the prosody component. We assume that the text analysis module has done a perfect job, and that the synthesis module does a perfect job, which cannot be done in general, so that approximations need to be made.

Evaluation can be done automatically or by using listening tests with human subjects. In both cases it's useful to start with some natural recordings with their associated text. We start by replacing the natural prosody with the system's synthetic prosody. In the case of automatic evaluation, we can compare the enriched prosodic representations described in Section 15.2 for both the natural recording and the synthetic prosody. The reference en-

riched prosodic representation can be obtained either manually or by using a pitch tracker and a speech recognizer.

Automated testing of prosody involves the following:

- **Duration.** It can be performed by measuring the average squared difference between each phone's actual duration in a real utterance and the duration predicted by the system.
- **Pitch** contours. It can be performed by using standard statistical measures over a system contour and a natural one. When this is done, duration and phoneme identity should be completely controlled. Measures such as root-mean-square error indicate the characteristic divergence between two contours, while correlation indicates the similarity in shape across difference pitch ranges. In general, RMSE scores of 15 Hz or less for male speech over a long sentence, with correlation of .8 or above, indicate quality that may be close to perceptually identical to the natural reference utterance. In general, such exactness of match is useful only during model training and testing and cannot be expected during training on entirely new utterances from random text.

Listening tests can be performed to evaluate a prosody module. This involves subjects listening to the natural recording and the synthetic speech, or to synthetic speech generated with two different prosody modules. This can lead to a more precise evaluation, as humans are the final consumer of this technology. However, such tests are more expensive to carry out. Furthermore, this method results in testing both the prosody module and the synthesis components together. To avoid this, the original waveform can be modified to have the synthetic prosody using the signal processing techniques described in Chapter 16. Since such techniques introduce some distortions, this measuring method is still somewhat biased. In practice, it has been shown that its effect is much smaller than that of the synthetic prosody [43].

It is shown that synthesizing pitch is more difficult than duration [43]. Subjects scored significantly higher utterances that had natural pitch and synthetic duration than utterances with synthetic pitch and natural duration. In fact, using only synthetic duration had a score quite close to that of the original recording. While duration modeling is not a solved problem, this indicates that generation of pitch contours is more difficult.

## 15.9. Historical Perspective and Further Reading

Prosodic methods have been incorporated within the traditional fields of rhetoric and elocution for centuries. In ancient Greece, at the time of Plato, written documentation in support of claims in legal disputes was rare. To help litigants plead their cases persuasively, systematic programs of rhetorical instruction were established, which included both content and form of verbal argument. This 'prescriptive' tradition of systematic instruction in verbal

style uncovered issues that remain central to the descriptively oriented prosodic research of today. A masterful and entertaining discussion of this tradition and its possible relevance to the task of teaching computers to *plead a case* can be found in [64]. The Greeks were particularly concerned about an issue that, as usual, is still important for us today: the separation of rhetorical effectiveness from considerations of truth. If you are interested in this, you cannot do better than to begin with Plato's dialog *Phaedrus* [42].

Modern linguists have also considered a related, but more narrowly formulated question: Should prosody be treated as a logical, categorical analog to phonological and syntactic processes? The best discussion of these issues from a prosodic (as opposed to strictly neurological) point of view is found in [7, 8]. If you are interested in the neurological side, you can begin with [13]. For emotional modeling, before slogging through the scattered and somewhat disjointed papers on emotion in speech that have appeared sporadically for years, the reader would be well advised to get a basic grounding in some of the issues related to emotion in computation, as treated in [38].

Going in the other direction, there are many subtle interactions in the phonetics of prosody: the various muscles, their joint possibilities of operation in phonation and articulation, as well as the acoustics properties of the vocal chambers. For an excellent introduction to the whole field, start with [27].

The most complete and accessible overview of modern prosodic analysis as embedded in mainstream linguistic theory is Ladd's *Intonational Phonology* [26], which covers the precursors, current standard practice, and remaining unsolved issues of the highly influential auto segmental theory of intonational phonology, from which ToBI has arisen. ToBI was devised by speech scientists who wanted a prosodic transcription standard to enable sharing of databases [4]. For most practical purposes, the ToBI definitions are sufficient as a starting point for both research and applications, but for those who prefer alternative annotation systems aligned with the British tradition, conversion guidelines have been attempted [45]. Another major phonological approach to English intonation has been the British school described in [11]. Bridging the two is IViE, a labeling system that is philosophically aligned with ToBI but may be more appropriate for non-U.S. dialects of English [18].

The first prosodic synthesis by rule was developed by Ignatius Mattingly in 1968 in Haskins Laboratories. In 1971, Fujisaki [15] developed his superposition model that has been used for many years. The development of the ToBI in 1992 [55] marked a milestone in automatic prosody generation. The application of statistical techniques, such as CART, for phoneme durations during the 1990s constituted a significant step beyond the rule-based methods. Finally, the development of the CHATR system in the mid-1990s ignited interest in the indexing of massive databases. It is possible to attempt smoothing over both the index space and the resulting prosodic data tracks by means of generalized learning methods, such as neural nets or HMMs. These models have built-in generalization over unseen inputs, and built-in smoothing over the concatenated outputs of unit selection. The network described in [57] codes every syllable in a training database in terms of perceived prominence (human judged), a number from 1 to 31, as well as the syllable's phonemes, rising/falling boundary type for phrase-edge syllables, and distance from preceding and following phrase bounda-

ries, for all syllables. When tested with reasonably simple text material of similar type, these networks yielded high-quality simulations.

A potential research area for future generalizations of this system is to increase the degree and accuracy of automation in labeling the training features of the recordings, such as perceived prominence. Another area is to either expand the inventory of model types, or to determine adequate generalization mechanisms. By training HMMs on accented syllables of differing phonetic structure, some of this fine alignment information can be automatically captured [16]. Another approach consists in generating pitch contours directly from a hidden Markov model, which is run in generation mode [66].

Recently, just as in speech synthesis for voice, there has been a realization that the *direct* and *parametric* prosodic models have a great deal in common. Direct models require huge databases of indexed exemplars for unmediated concatenation and playback of contours, in addition to generalized back-off methods, while parametric models are generalized for any input, but also require phonetic databases of sufficient variety to support statistical learning of parameter settings for high quality. We can, therefore, expect to see increasing numbers of hybrid systems. One such system is described in [47], which could be viewed as a stochastic realization of Fujisaki's superposition model without the phrase controls, where the accents are given by ToBI labels and the smoothing is done by means of a dynamical system.

While this chapter has focused on U.S. English, many similar issues arise in prosodic modeling of other languages. An excellent survey of the prosodic systems of every major European language, as well as Arabic and several major East Asian languages, can be found in [21].

Though not explicitly covered in this chapter, analysis of prosody for speech recognition is a small but growing area of study. Anyone who has digested this chapter should be prepared to approach the more specialized work of [25, 37] and the speech recognition prosody studies collected in [48]. Those with a psycholinguistic bent can begin with [12].

## REFERENCES


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

[2] Anderson, M.D., J.B. Pierrehumbert, and M.Y. Liberman, "Synthesis by Rule of English Intonation Patterns," *Proc. of Int. Conf. on Acoustics, Speech and Signal Processing*, 1984, pp. 2.8.1-2.8.4.

[3] Arnfield, S., "Word Class Driven Synthesis of Prosodic Annotations," *Proc. of the Int. Conf. on Spoken Language Processing*, 1996, Philadelphia, PA, pp. 1978-1981.

[4] Beckman, M.E. and G.M. Ayers, *Guidelines for ToBI Labelling*, 1994, http://www.ling.ohio-state.edu/phonetics/ToBI/main.html.

[5] Black, A. and A. Hunt, "Generating F0 Contours from ToBI labels using Linear Regression," *Proc. of the Int. Conf. on Spoken Language Processing*, 1996, pp. 1385-1388.

[6] Bolinger, D., "Accent is predictable (if you're a mind-reader)," *Language*, 1972, **48**, pp. 633-44.

[7] Bolinger, D., *Intonation and its parts*, 1986, Stanford, Stanford University Press.

[8] Bolinger, D., *Intonation and its uses*, 1989, Stanford, Stanford University Press.

[9] Campbell, N., "CHATR: A High-Definition Speech Re-sequencing System," *ASA/ASJ Joint Meeting*, 1996, Honolulu, Hawaii, pp. 1223-1228.

[10] Carroll, L., *Alice in Wonderland, Unabridged ed.*, 1997, Penguin USA.

[11] Crystal, D., *Prosodic Systems and Intonation in English*, 1969, Cambridge University Press.

[12] Crystal, D., "Prosody and Parsing," P. Warren, Editor, 1996, Lawrence Erlbaum Associates.

[13] Emmorey, K., "The Neurological Substrates for Prosodic Aspects of Speech," *Brain and Language*, 1987, **30**, pp. 305-320.

[14] Fujisaki, H., "Prosody, Models, and Spontaneous Speech" in *Computing Prosody*, Y. Sagisaka, N. Campbell, N. Higuchi, Editors, 1997, New York, Springer.

[15] Fujisaki, H. and H. Sudo, "A Generative Model of the Prosody of Connected Speech in Japanese," *Annual Report of Eng. Research Institute*, 1971, **30**, pp. 75-80.

[16] Fukada, T., *et al.*, "A Study on Pitch Pattern Generation Using HMM-based Statistical Information," *Proc. Int. Conf. on Spoken Language Processing*, 1994, Yokohama, Japan, pp. 723-726.

[17] Goldsmith, J., "English as a Tone Language" in *Phonology in the 1980's*, D. Goyvaerts, Editor 1980, Ghent, Story-Scientia.

[18] Grabe, E., F. Nolan, and K. Farrar, "IViE - a Comparative Transcription System for Intonational Variation in English," *Proc. of the Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia.

[19] Hirose, H. and H. Fujisaki, "Analysis and Synthesis of Voice Fundamental Frequency Contours of Spoken Sentences," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1982, pp. 950-953.

[20] Hirschberg, J., "Pitch Accent in Context: Predicting Intonational Prominence from Text," *Artificial Intelligence*, 1993, **63**, pp. 305-340.

[21] Hirst, D., A.D. Cristo, and A. Cruttenden, *Intonation Systems: A Survey of Twenty Languages*, 1998, Cambridge, U.K., Cambridge University Press.

[22] Hirst, D.J., "The Symbolic Coding of Fundamental Frequency Curves: from Acoustics to Phonology," *Proc. of Int. Symposium on Prosody*, 1994, Yokohama, Japan.

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

[24] Klasmeyer, G. and W.F. Sendlmeier, "The Classification of Different Phonation Types in Emotional and Neutral Speech," *Forensic Linguistics*, 1997, **4**(1), pp. 104-125.

[25] Kompe, R., *Prosody in Speech Understanding Systems*, 1997, Berlin, Springer.

[26] Ladd, R.D., *Intonational Phonology*, Cambridge Studies in Linguistics, 1996, Cambridge, Cambridge University Press.

[27] Ladefoged, P., *A Course in Phonetics*, 1993, Harcourt Brace Jovanovich.

[28] Liberman, M., *The Intonation System of English*, PhD Thesis in Linguistics and Philosophy, 1975, MIT, Cambridge.

[29] Liberman, M. and J. Pierrehumbert, "Intonational Invariance under Changes in Pitch Range and Length" in *Language and Sound Structure*, M. Aronoff, Oerhle, R., ed., 1984, Cambridge, MA, MIT Press, pp. 157-233.

[30] Lopez-Gonzalo, E., and J.M. Rodriguez-Garcia, "Statistical Methods in Data-Driven Modeling of Spanish Prosody for Text to Speech," *in Proc. ICSLP 1996*, 1996, pp. 1373-1376.

[31] Malfrere, F., T. Dutoit, and P. Mertens, "Automatic Prosody Generation Using Supra-Segmental Unit Selection," *Third ESCA/COCOSCA Workshop on Speech Synthesis*, 1998, Jenolan Caves, Australia, pp. 323-328.

[32] Mason, J., *An Essay on Elocution*, 1st ed, 1748, London.

[33] Möbius, B., "Analysis and Synthesis of German F0 Contours by Means of Fujisaki's Model," *Speech Communication*, 1993, **13**(53-61).

[34] Murray, I. and J. Arnott, "Toward the Simulation of Emotion in Synthetic Speech: A Review of the Literature on Human Vocal Emotion," *Journal Acoustical Society of America*, 1993, **93**(2), pp. 1097-1108.

[35] Öhman, S., *Word and Sentence Intonation: A Quantitative Model*, 1967, KTH, pp. 20-54.

[36] Ostendorf, M., and N. Veilleux, "A Hierarchical Stochastic Model for Automatic Prediction of Prosodic Boundary Location," *Computational Linguistics*, 1994, **20**(1), pp. 27-54.

[37] Ostendorf, M., "Linking Speech Recognition and Language Processing Through Prosody," *Journal for the Integrated Study of Artificial Intelligence, Cognitive Science and Applied Epistemology*, 1998, **15**(3), pp. 279-303.

[38] Picard, R.W., *Affective Computing*, 1997, MIT Press.

[39] Pierrehumbert, J., *The Phonology and Phonetics of English Intonation*, PhD Thesis in *Linguistics and Philosophy* 1980, MIT, Cambridge, MA.

[40] Pierrehumbert, J., and M. Beckman, *Japanese Tone Structure*, 1988, Cambridge, MA, MIT Press.

[41] Pierrehumbert, J. and J. Hirschberg, "The Meaning of Intonational Contours in the Interpretation of Discourse" in *Intentions in Communication*, P.R. Cohen, J. Morgan, and M. E. Pollack, ed., 1990, Cambridge, MA, MIT Press.

[42] Plato, *The Symposium and The Phaedrus: Plato's Erotic Dialogues*, 1994, State University of New York Press.

[43] 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.

[44] Prevost, S. and M. Steedman, "Specifying Intonation from Context for Speech Synthesis," *Speech Communication*, 1994, **15**, pp. 139-153.

[45] Roach, P., "Conversion between Prosodic Transcription Systems: 'Standard British' and ToBI," *Speech Communication*, 1994, **15**, pp. 91-97.

[46] Ross, K. and M. Ostendorf, "Prediction of Abstract Prosodic Labels for Speech Synthesis," *Computer, Speech and Language*, 1996, **10**, pp. 155-185.

[47] Ross, K. and M. Ostendorf, "A Dynamical System Model for Generating Fundamental Frequency for Speech Synthesis," *IEEE Trans. on Speech and Audio Processing*, 1999, 7(3), pp. 295-309.

[48] Sagisaka, Y., W.N. Campbell, and N. Higuchi, *Computing Prosody*, 1997, Springer-Verlag.

[49] Santen, J.V., "Contextual Effects on Vowel Duration," *Speech Communication*, 1992, **11**(6), pp. 513-546.

[50] Selting, M., *Prosodie im Gespräch*, 1995, Max Niemeyer Verlag.

[51] Shattuck-Hufnagel, S. and M. Ostendorf, "Stress Shift and Early Pitch Accent Placement in Lexical Items in American English," *Journal of Phonetics*, 1994, **22**, pp. 357-388.

[52] Shen, X.-n.S., *The Prosody of Mandarin Chinese*, 1990, Berkeley, University of California Press.

[53] Sheridan, T., *Lectures on the Art of Reading*, 3rd ed, 1787, London, Dodsley.

[54] Silverman, K., *The Structure and Processing of Fundamental Frequency Contours*, Ph.D. Thesis, 1987, University of Cambridge, Cambridge, UK.

[55] Silverman, K., "ToBI: A Standard for Labeling English Prosody," *Int. Conf. on Spoken Language Processing*, 1992, Banff, Canada, pp. 867-870.

[56] Sokal, A.D., "Transgressing the Boundaries: Towards a Transformative Hermeneutics of Quantum Gravity," *Social Text*, 1996, **46/47**, pp. 217-252.

[57] Sonntag, G., T. Portele, and B. Heuft, "Prosody Generation with a Neural Network: Weighing the Importance of Input Parameters," *Proc. Int. Conf. on Acoustics, Speech and Signal Processing*, 1997, pp. 930-934.

[58] Steedman, M., "Information Structure and the Syntax-Phonology Interface," *Linguistic Inquiry*, 2000.

[59] Stevens, K.N., "Control Parameters for Synthesis by Rule," *Proc. of the ESCA Tutorial Day on Speech Synthesis*, 1990, pp. 27-37.

[60] Taylor, P.A., "The Tilt Intonation Model," *Proc. Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia.

[61] van Santen, J., "Assignment of Segmental Duration in Text-to-Speech Synthesis," *Computer Speech and Language*, 1994, **8**, pp. 95-128.

[62] van Santen, J., "Segmental Duration and Speech Timing" in *Computing Prosody*, Y. Sagisaka, N. Campbell, and N. Higuchi, eds., 1997, New York, Springer, pp. 225-250.

[63] van Santen, J. and J. Hirschberg, "Segmental Effects of Timing and Height of Pitch Contours," *Proc. of the Int. Conf. on Spoken Language Processing*, 1994, pp. 719-722.

[64] Vanderslice, R.L., *Synthetic Elocution: Considerations in Automatic Orthographic-to-Phonetic Conversion of English with Special Reference to Prosody*, PhD Thesis, 1968, UCLA, Los Angeles.

[65] W3C, *Speech Synthesis Markup Requirements for Voice Markup Languages*, 2000, http://www.w3.org/TR/voice-tts-reqs/.

[66] Yoshimura, T., *et al.*, "Simultaneous Modeling of Spectrum, Pitch and Duration in HMM-Based Speech Synthesis," *EuroSpeech*, 1999, Budapest, Hungary, pp. 2347-2350.

# CHAPTER 16

# Speech Synthesis

The speech synthesis module of a TTS system is the component that generates the waveform. The input of traditional speech synthesis components is a phonetic transcription with its associated prosody. The input can also include the original text with tags, as this may help in producing higher-quality speech.

Speech synthesis can be classified into three types according to the model used in the speech generation. *Articulatory synthesis*, described in Section 16.2.4, uses a physical model of speech production that includes all the articulators described in Chapter 2. *Formant synthesis* uses a source-filter model, where the filter is characterized by slowly varying formant frequencies; it is the subject of Section 16.2. *Concatenative synthesis* generates speech by concatenating speech segments and is described in Section 16.3. To allow more flexibility in concatenative synthesis, a number of prosody modification techniques are described in Sections 16.4 and 16.5. Finally, a guide to evaluating speech synthesis systems is included in Section 16.6.

Speech synthesis can also be classified according to the degree of manual intervention in the system design into *synthesis by rule* and *data-driven synthesis*. In the former, a set of manually derived rules is used to drive a synthesizer, and in the latter the synthesizer's parameters are obtained automatically from real speech data. Concatenative systems are, thus, data driven. Formant synthesizers have traditionally used synthesis by rule, since the evolution of formants in a formant synthesizer has been done with hand-derived rules. Nonetheless, formant transitions can also be trained from data, as we show in Section 16.2.3.

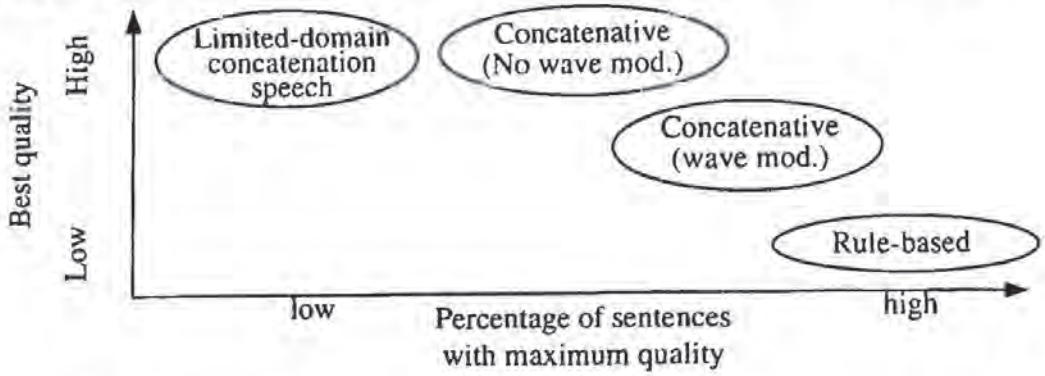



**Figure 16.1** Quality and task-independence in speech synthesis approaches.

## 16.1. ATTRIBUTES OF SPEECH SYNTHESIS

The most important attribute of a speech synthesis system is the quality of its output speech. It is often the case that a single system can sound beautiful on one sentence and terrible on the next. For that reason we need to consider the quality of the best sentences and the percentage of sentences for which such quality is achieved. This tradeoff is illustrated in Figure 16.1, where we compare four different families of speech generation approaches:

- *Limited-domain waveform concatenation.* For a given limited domain, this approach can generate very high quality speech with only a small number of recorded segments. Such an approach, used in most interactive voice response systems, cannot synthesize arbitrary text. Many concept-to-speech systems, described in Chapter 17, use this approach.
- *Concatenative synthesis with no waveform modification.* Unlike the previous approach, these systems can synthesize speech from arbitrary text. They can achieve good quality on a large set of sentences, but the quality can be mediocre for many other sentences where poor concatenations take place.

- *Concatenative systems with waveform modification.* These systems have more flexibility in selecting the speech segments to concatenate because the waveforms can be modified to allow for a better prosody match. This means that the number of sentences with mediocre quality is lower than in the case where no prosody modification is allowed. On the other hand, replacing natural with synthetic prosody can hurt the overall quality. In addition, the prosody modification process also degrades the overall quality.
- *Rule-based systems.* Such systems tend to sound uniformly across different sentences, albeit with quality lower than the best quality obtained in the systems above.

Best-quality and quality variability are possibly two of the most important attributes of a speech synthesis system, but not the only ones. Measuring quality, difficult to do in an objective way, is the main subject of Section 16.6. Other attributes of a speech synthesis system include:

- *Delay.* The time it takes for the synthesizer to start speaking is important for interactive applications and should be less than 200 ms. This delay is composed of the algorithmic delays of the front end and of the speech synthesis module, as well as the computation involved.
- *Memory resources.* Rule-based synthesizers require, on the average, less than 200 KB, so they are a widely used option whenever memory is at a premium. However, required RAM can be an issue for concatenative systems, some of which may require over 100 MB of storage.
- *CPU resources.* With current CPUs, processing time is typically not an issue, unless many channels need to run in the same CPU. Nonetheless, some concatenative synthesizers may require a large amount of computation when searching for the optimal sequence.
- *Variable speed.* Some applications may require the speech synthesis module to generate variable speed, particularly fast speech. This is widely used by blind people who need TTS systems to obtain their information and can accept fast speech because of the increased throughput. Fast speech is also useful when skimming material. Concatenative systems that do not modify the waveform cannot achieve variable speed control, unless a large number of segments are recorded at different speeds.
- *Pitch control.* Some spoken language systems require the output speech to have a specific pitch. This is the case if you want to generate voice for a song. Again, concatenative systems that do not modify the waveform cannot do this, unless a large number of speech segments are recorded at different pitch.

- *Voice characteristics.* Other spoken language systems require specific voices, such as that of a robot, that cannot be recorded naturally, or some, such as monotones, that are tedious to record. Since rule-based systems are so flexible, they are able to do many such modifications.

The approaches described in this chapter assume as input a phonetic string, durations, a pitch contour, and possibly volume. Pauses are signaled by the default phoneme SIL with its corresponding duration. If the parsed text is available, it is possible to do even better in a concatenative system by conducting a matching with all the available information.

## 16.2. FORMANT SPEECH SYNTHESIS

As discussed in Chapter 6, we can synthesize a stationary vowel by passing a glottal periodic waveform through a filter with the formant frequencies of the vocal tract. For the case of unvoiced speech we can use white random noise as the source instead. In practice, speech signals are not stationary, and we thus need to change the pitch of the glottal source and the formant frequencies over time. The so-called *synthesis-by-rule* refers to a set of rules on how to modify the pitch, formant frequencies, and other parameters from one sound to another while maintaining the continuity present in physical systems like the human production system. Such a system is described in the block diagram of Figure 16.2.

In Section 16.2.1 we describe the second block of Figure 16.2, the formant synthesizer that generates a waveform from a set of parameters. In Section 16.2.2 we describe the first block of Figure 16.2, the set of rules that can generate such parameters. This approach was the one followed by Dennis Klatt and his colleagues [4, 30]. A data-driven approach to this first block is studied in Section 16.2.3. Finally, articulatory synthesis is the topic of Section 16.2.4.

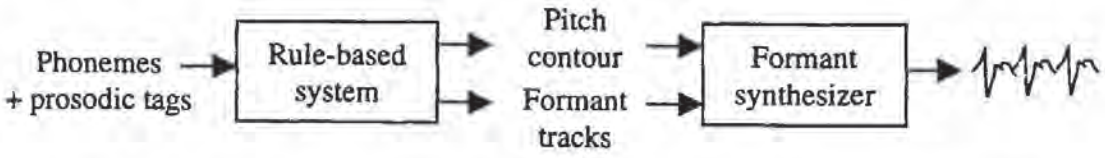



**Figure 16.2** Block diagram of a synthesis-by-rule system. Pitch and formants are listed as the only parameters of the synthesizer for convenience. In practice, such system has about 40 parameters.

### 16.2.1. Waveform Generation from Formant Values

To be able to synthesize speech by rule, a simple model for the filter is needed. Most rule-based synthesizers use the so-called *formant synthesis*, which is derived from models of speech production. The model explicitly represents a number of formant resonances (from 2 to 6). A formant resonance can be implemented (see Chapter 6) with a second-order IIR filter

$$H_i(z) = \frac{1}{1 - 2e^{-\pi b_i} \cos(2\pi f_i) z^{-1} + e^{-2\pi b_i} z^{-2}} \tag{16.1}$$

with $f_i = F_i / F_s$ and $b_i = B_i / F_s$, where $F_i$, $B_i$, and $F_s$ are the formant's center frequency, formant's bandwidth, and sampling frequency, respectively, all in Hz. A filter with several resonances can be constructed by cascading several such second-order sections (*cascade model*) or by adding several such sections together (*parallel model*). Formant synthesizers typically use the parallel model to synthesize fricatives and stops and the cascade model for all voiced sounds.

Unlike the cascade model, the parallel model requires gains to be specified for each second-order section, which often are chosen proportional to the formant's frequency and inversely proportional to the formant's bandwidth. The cascade model results in an all-pole filter, whereas the parallel model has zeros in addition to poles. Such a combination is shown in Figure 16.3, where R1 through R6 are the resonances 1 to 6 and each one represents a second-order IIR filter like that in Eq. (16.1). RNP represents the nasal resonance, and RNZ is an FIR filter with the nasal zero. A1 through AB are the gains for each filter, used for the parallel model. Switch SW controls whether the cascade model or parallel model is used.

For voiced sounds the excitation model consists of an impulse train driving a low-pass filter RGP and then a bandpass filter created by the parallel of RGZ and RGS. For unvoiced sounds the excitation consists of white noise driving a low-pass filter LPF. The excitation for voiced fricatives is a combination of the two sources. In practice, this mixed excitation is used for all voiced sounds to add some breathiness. Klatt [30] showed that this model could reproduce quite faithfully a natural recording if the parameters had been manually selected.

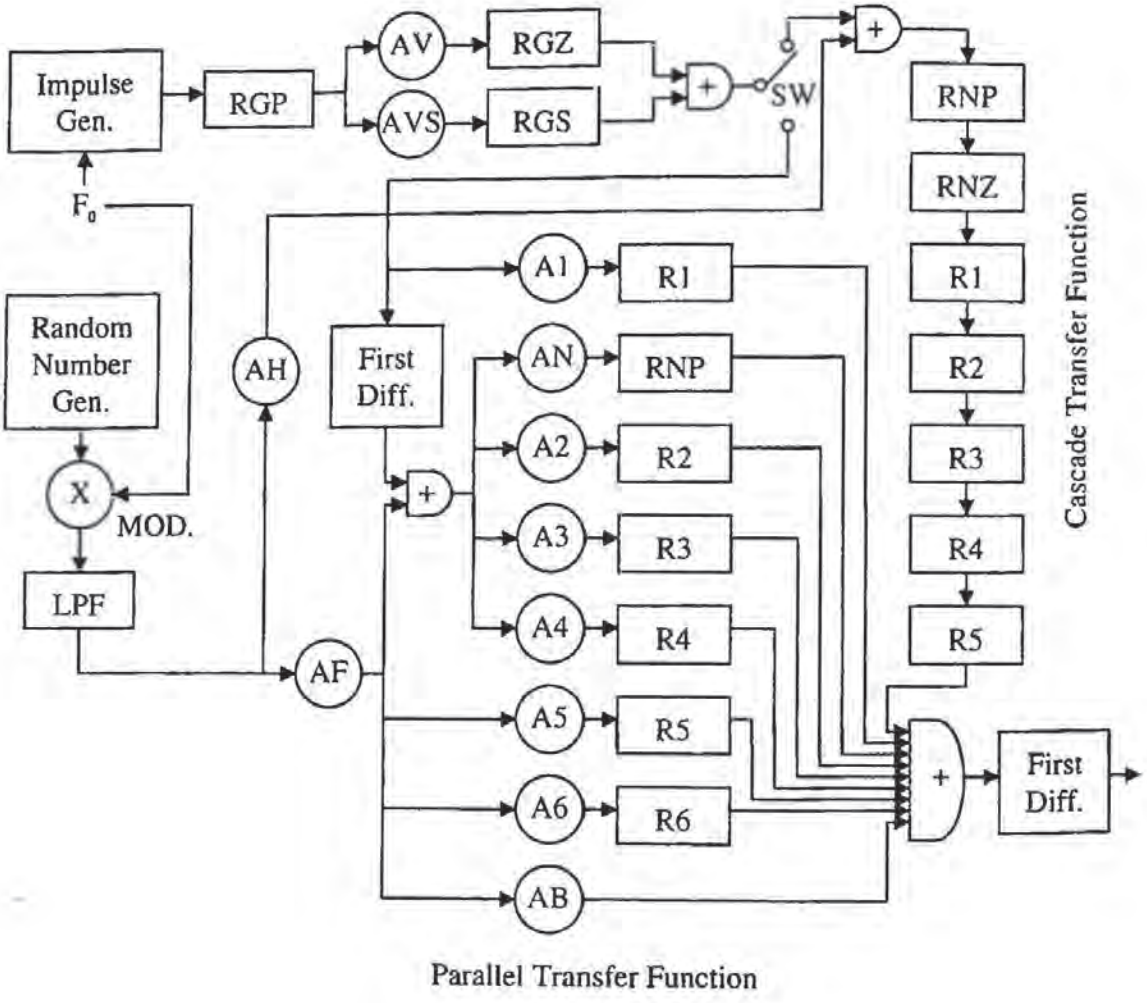



**Figure 16.3** Block diagram of the Klatt formant synthesizer (after Allen [4]).

The parameter names and their minimum and maximum values are listed in Table 16.1, where the switch SW can be in voiced (V) or consonant (C) mode. For example, in Figure 16.3, R2 is the resonator corresponding to the second formant, whose center frequency F2 and bandwidth B2 are given in Table 16.1. In addition to the six resonances associated to the six formants, there are other resonances: RGP, RGZ, RGS, RNP, and RNZ. Other source models are also possible [43].

Table 16.1 Parameter values for Klatt's cascade/parallel formant synthesizer with the parameter symbol, full name, minimum, maximum, and typical values (after Allen [4]).

| N | Symbol | Name | Min | Max | Typ |
|---|---|---|---|---|---|
| 1 | AV | Amplitude of voicing (dB) | 0 | 80 | 0 |
| 2 | AF | Amplitude of frication (dB) | 0 | 80 | 0 |
| 3 | AH | Amplitude of aspiration (dB) | 0 | 80 | 0 |
| 4 | AVS | Amplitude of sinusoidal voicing (dB) | 0 | 80 | 0 |
| 5 | F0 | Fundamental frequency (Hz) | 0 | 500 | 0 |
| 6 | F1 | First formant frequency (Hz) | 150 | 900 | 500 |
| 7 | F2 | Second formant frequency (Hz) | 500 | 2500 | 1500 |
| 8 | F3 | Third formant frequency (Hz) | 1300 | 3500 | 2500 |
| 9 | F4 | Fourth formant frequency (Hz) | 2500 | 4500 | 3500 |
| 10 | FNZ | Nasal zero frequency (Hz) | 200 | 700 | 250 |
| 11 | AN | Nasal formant amplitude (Hz) | 0 | 80 | 0 |
| 12 | A1 | First formant amplitude (Hz) | 0 | 80 | 0 |
| 13 | A2 | Second formant amplitude (Hz) | 0 | 0 | 0 |
| 14 | A3 | Third formant amplitude (Hz) | 0 | 80 | 0 |
| 15 | A4 | Fourth formant amplitude (Hz) | 0 | 80 | 0 |
| 16 | A5 | Fifth formant amplitude (Hz) | 0 | 80 | 0 |
| 17 | A6 | Sixth formant amplitude (Hz) | 0 | 80 | 0 |
| 18 | AB | Bypass path amplitude (Hz) | 0 | 80 | 0 |
| 19 | B1 | First formant bandwidth (Hz) | 40 | 500 | 50 |
| 20 | B2 | Second formant bandwidth (Hz) | 40 | 500 | 70 |
| 21 | B3 | Third formant bandwidth (Hz) | 40 | 500 | 110 |
| 22 | SW | Cascade/parallel switch | 0 | 1 | 0 |
| 23 | FGP | Glottal resonator 1 frequency (Hz) | 0 | 600 | 0 |
| 24 | BGP | Glottal resonator 1 bandwidth (Hz) | 100 | 2000 | 100 |
| 25 | FGZ | Glottal zero frequency (Hz) | 0 | 500 | 1500 |
| 26 | BGZ | Glottal zero bandwidth (Hz) | 100 | 9000 | 6000 |
| 27 | B4 | Fourth formant bandwidth (Hz) | 100 | 500 | 250 |
| 28 | F5 | Fifth formant frequency (Hz) | 3500 | 4900 | 3850 |
| 29 | B5 | Fifth formant bandwidth (Hz) | 150 | 700 | 200 |
| 30 | F6 | Sixth formant frequency (Hz) | 4000 | 4999 | 4900 |
| 31 | B6 | Sixth formant bandwidth (Hz) | 200 | 2000 | 100 |
| 32 | FNP | Nasal pole frequency (Hz) | 200 | 500 | 250 |
| 33 | BNP | Nasal pole bandwidth (Hz) | 50 | 500 | 100 |
| 34 | BNZ | Nasal zero bandwidth (Hz) | 50 | 500 | 100 |
| 35 | BGS | Glottal resonator 2 bandwidth (Hz) | 100 | 1000 | 200 |
| 36 | SR | Sampling rate (Hz) | 500 | 20000 | 10000 |
| 37 | NWS | Number of waveform samples per chunk | 1 | 200 | 50 |
| 38 | G0 | Overall gain control (dB) | 0 | 80 | 48 |
| 39 | NFC | Number of cascaded formants | 4 | 6 | 5 |

## 16.2.2. Formant Generation by Rule

As described in Chapter 2, formants are one of the main features of vowels. Because of the physical limitations of the vocal tract, formants do not change abruptly with time. Rule-based formant synthesizers enforce this by generating continuous values for $f_i[n]$ and $b_i[n]$, typically every 5–10 milliseconds. Continuous values can be implemented through the above structures if a lattice filter is used, because it allows its reflection coefficients to vary at every sample (see Chapter 6). In practice, the values can be fixed within a frame as long as frames are smaller than 5 ms.

Rules on how to generate formant trajectories from a phonetic string are based on the *locus theory* of speech production. The locus theory specifies that formant frequencies within a phoneme tend to reach a stationary value called the *target*. Targets for formant frequencies and bandwidths for a male speaker are shown in Table 16.2 (nonvocalic segments) and Table 16.3 (vocalic segments). This target is reached if either the phoneme is sufficiently long or the previous phoneme's target is close to the current phoneme's target. The maximum slope at which the formants move is dominated by the speed of the articulators, determined by physical constraints. Since each formant is primarily caused by the position of a given articulator, formants caused by the body of the tongue do not vary as rapidly as formants caused by the tip of the tongue or the lips. Thus, rule-based systems store targets for each phoneme as well as maximum allowable slopes and transition times.

For example, a transition between a vowel and a sonorant can follow the rule shown in Figure 16.4 with $a_1$ being the target of unit 1 and $a_2$ the target of unit 2. The values of $T_{cb}$ and $T_{cf}$ are 40 and 80 ms, respectively, and $a_b = a_2 + 0.75(a_1 - a_2)$. The time $T_{cb} + T_{cf}$ specifies how rapid the transition is. While linear interpolation could be used, $a_b$ and the ratio $T_{cb}/T_{cf}$ are sometimes used to further refine the shape of the formant transition.

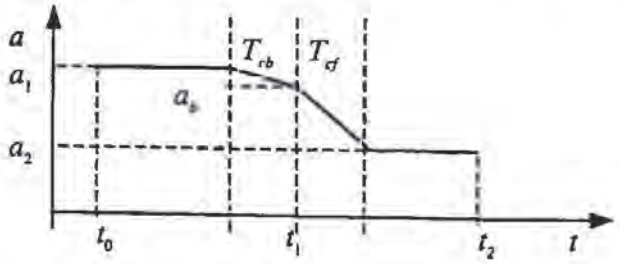



**Figure 16.4** Transition between two vowels in a formant synthesizer.

Other rules can allow a discontinuity, for example, when a transition out of an unvoiced segment takes place. To improve naturalness, all these parameters can be made dependent on the immediate phonetic context.

**Table 16.2** Targets used in the Klatt synthesizer: formant frequencies and bandwidths for non-vocalic segments of a male speaker. Note that in addition to the phoneme set used in Chapter 2, there are several additional phonetic segments here such as *axp, dx, el, em, en, gp, hx, kp, lx, qq, rx, tq, wh*, that allow more control (after Allen [4]).

| | F1 | F2 | F3 | B1 | B2 | B3 |
|---|---|---|---|---|---|---|
| *axp* | 430 | 1500 | 2500 | 120 | 60 | 120 |
| *b* | 200 | 900 | 2100 | 65 | 90 | 125 |
| *ch* | 300 | 1700 | 2400 | 200 | 110 | 270 |
| *d* | 200 | 1400 | 2700 | 70 | 115 | 180 |
| *dh* | 300 | 1150 | 2700 | 60 | 95 | 185 |
| *dx* | 200 | 1600 | 2700 | 120 | 140 | 250 |
| *el* | 450 | 800 | 2850 | 65 | 60 | 80 |
| *em* | 200 | 900 | 2100 | 120 | 60 | 70 |
| *en* | 200 | 1600 | 2700 | 120 | 70 | 110 |
| *ff* | 400 | 1130 | 2100 | 225 | 120 | 175 |
| *g* | 250 | 1600 | 1900 | 70 | 145 | 190 |
| *gp* | 200 | 1950 | 2800 | 120 | 140 | 250 |
| *h* | 450 | 1450 | 2450 | 300 | 160 | 300 |
| *hx* | 450 | 1450 | 2450 | 200 | 120 | 200 |
| *j* | 200 | 1700 | 2400 | 50 | 110 | 270 |
| *k* | 350 | 1600 | 1900 | 280 | 220 | 250 |
| *kp* | 300 | 1950 | 2800 | 150 | 140 | 250 |
| *l* | 330 | 1050 | 2800 | 50 | 100 | 280 |
| *lx* | 450 | 800 | 2850 | 65 | 60 | 80 |
| *m* | 480 | 1050 | 2100 | 40 | 175 | 120 |
| *ng* | 480 | 1600 | 2050 | 160 | 150 | 100 |
| *n* | 480 | 1400 | 2700 | 40 | 300 | 260 |
| *p* | 300 | 900 | 2100 | 300 | 190 | 185 |
| *qq* | 400 | 1400 | 2450 | 120 | 140 | 250 |
| *r* | 330 | 1060 | 1380 | 70 | 100 | 120 |
| *rx* | 460 | 1260 | 1560 | 60 | 60 | 70 |
| *sh* | 400 | 1650 | 2400 | 200 | 110 | 280 |
| *sil* | 400 | 1400 | 2400 | 120 | 140 | 250 |
| *s* | 400 | 1400 | 2700 | 200 | 95 | 220 |
| *th* | 400 | 1150 | 2700 | 225 | 95 | 200 |
| *tq* | 200 | 1400 | 2700 | 120 | 140 | 250 |
| *t* | 300 | 1400 | 2700 | 300 | 180 | 220 |
| *v* | 300 | 1130 | 2100 | 55 | 95 | 125 |
| *wh* | 330 | 600 | 2100 | 150 | 60 | 60 |
| *w* | 285 | 610 | 2150 | 50 | 80 | 60 |
| *y* | 240 | 2070 | 3020 | 40 | 250 | 500 |
| *zh* | 300 | 1650 | 2400 | 220 | 140 | 250 |
| *z* | 300 | 1400 | 2700 | 70 | 85 | 190 |

**Table 16.3** Targets used in the Klatt synthesizer: formant frequencies and bandwidths for vocalic segments of a male speaker. Note that in addition to the phoneme set used in Chapter 2, there are several additional phonetic segments here such as *axr, exr, ix, ixr, oxr, uxr, yu* that allow more control (after Allen [4]).

| | F1 | F2 | F3 | B1 | B2 | B3 |
|---|---|---|---|---|---|---|
| *aa* | 700 | 1220 | 2600 | 130 | 70 | 160 |
| *ae* | 620 | 1660 | 2430 | 70 | 130 | 300 |
| *ah* | 620 | 1220 | 2550 | 80 | 50 | 140 |
| *ao* | 600 | 990 | 2570 | 90 | 100 | 80 |
| *aw* | 640 | 1230 | 2550 | 80 | 70 | 110 |
| *ax* | 550 | 1260 | 2470 | 80 | 50 | 140 |
| *axr* | 680 | 1170 | 2380 | 60 | 60 | 110 |
| *ay* | 660 | 1200 | 2550 | 100 | 120 | 200 |
| *eh* | 530 | 1680 | 2500 | 60 | 90 | 200 |
| *er* | 470 | 1270 | 1540 | 100 | 60 | 110 |
| *exr* | 460 | 1650 | 2400 | 60 | 80 | 140 |
| *ey* | 480 | 1720 | 2520 | 70 | 100 | 200 |
| *ih* | 400 | 1800 | 2670 | 50 | 100 | 140 |
| *ix* | 420 | 1680 | 2520 | 50 | 100 | 140 |
| *ixr* | 320 | 1900 | 2900 | 70 | 80 | 120 |
| *iy* | 310 | 2200 | 2960 | 50 | 200 | 400 |
| *ow* | 540 | 1100 | 2300 | 80 | 70 | 70 |
| *oxr* | 550 | 820 | 2200 | 60 | 60 | 60 |
| *oy* | 550 | 960 | 2400 | 80 | 120 | 160 |
| *uh* | 450 | 1100 | 2350 | 80 | 100 | 80 |
| *uw* | 350 | 1250 | 2200 | 65 | 110 | 140 |
| *uxr* | 360 | 800 | 2000 | 60 | 60 | 80 |
| *yu* | 290 | 1900 | 2600 | 70 | 160 | 220 |

Klatt showed that for a given natural utterance, he could manually obtain a sequence of formant tracks $f_i[n]$ and $b_i[n]$, such that the synthesized utterance not only had good quality but also sounded very similar to the original. This shows that the formant synthesizer of Section 16.2.1 appears to be sufficient for generation. On the other hand, when the formant tracks are obtained automatically through rules such as that of Figure 16.4 and Table 16.2 and Table 16.3, the output speech does not sound that natural, and the voice does not resemble the voice of the original recording.

Formant synthesis is very flexible because it can generate intelligible speech with relatively few parameters (about 40). The use of context-dependent rules can improve the quality of the synthesizer at the expense of a great deal of manual tuning. The synthesized speech is, by design, smooth, although it may not resemble any given speaker and may not sound very natural.

Because of their flexibility, formant synthesizers can often generate many different voices and effects. While not as flexible, voice effects can also be produced in concatenative speech systems (see Section 16.5.3).

### 16.2.3. Data-Driven Formant Generation

While, in general, formant synthesis assumes the formant model of Section 16.2.1 driven by parameter values generated by rules, as in Section 16.2.2, data-driven methods to generate the formant values have also been proposed [3]. An HMM running in generation mode emits three formant frequencies and their bandwidths every 10 ms, and these values are used in a cascade formant synthesizer similar to that described in Section 16.2.1. Like the speech recognition counterparts, this HMM has many decision-tree context-dependent triphones and three states per triphone. A Gaussian distribution per state is used in this work. The baseline system uses a six-dimensional vector that includes the first three formant frequencies and their bandwidths. Initially it is assumed that the input to the synthesizer includes, in addition to the duration of each phoneme, the duration of each state. In this case, the maximum likelihood formant track is a sequence of the state means and, therefore, is discontinuous at state boundaries.

The key to obtaining a smooth formant track is to augment the feature vector with the corresponding delta formants and bandwidths (the difference between the feature at time $t$ and that feature at time $t - 1$) to complete a twelve-dimensional vector. The maximum likelihood solution now entails solving a tridiagonal set of linear equations (see the discussion on statistical formant tracking in Chapter 6). The resulting formant track is smooth, as it balances formant values that are close to the state means with delta values that are also within the state means. In addition, the synthesized speech resembles that of the donor speaker. More details on the analysis and model training can be found in the formant tracking section of Chapter 6.

### 16.2.4. Articulatory Synthesis

Articulatory synthesis is another model that has been used to synthesize speech by rule, by using parameters that model the mechanical motions of the articulators and the resulting distributions of volume velocity and sound pressure in the lungs, larynx, and vocal and nasal tracts. Because the human speech production articulators do not have that many degrees of freedom, articulatory models often use as few as 15 parameters to drive a formant synthesizer.

The relationship between articulators and acoustics is many-to-one [17]. For example, a ventriloquist can produce speech sounds with very different articulator positions than those of normal speech. The *speech inversion* problem is therefore ill posed. By using the assumption that the articulators do not change rapidly over time, it is possible, however, to estimate

the vocal-tract area from formant frequencies [37]. In [8] the model uses five articulatory parameters: area of lip opening, constriction formed by the tongue blade, opening to the nasal cavities, average glottal area, and rate of active expansion or contraction of the vocal tract volume behind a constriction. These five parameters are augmented with the first four formant frequencies and F0.

Those area parameters can be obtained from real speech through X-rays and magnetic resonance imaging (MRI), though positioning such sensors in the vocal tract alters the way speech is produced (such as the sensors in the lips) and impedes completely natural sounds. The relationship between articulatory parameters and acoustic values has typically been done using a nonlinear mapping such as a neural network or a codebook.

While one day this may be the best way to synthesize speech, the state-of-the-art in articulatory synthesis does not generate speech with quality comparable to that of formant or concatenative systems.

## 16.3. CONCATENATIVE SPEECH SYNTHESIS

While state-of-the-art synthesis by rule is quite intelligible, it sounds unnatural, because it is very difficult to capture all the nuances of natural speech in a small set of manually derived rules. In *concatenative synthesis*, a speech segment is synthesized by simply playing back a waveform with matching phoneme string. An utterance is synthesized by concatenating together several speech fragments. The beauty of this approach is that unlike synthesis-by-rule, it requires neither rules nor manual tuning. Moreover, each segment is completely natural, so we should expect very natural output.

Unfortunately, this is equivalent to assembling an automobile with parts of different colors: each part is very good yet there is a color discontinuity from part to part that makes the whole automobile unacceptable. Speech segments are greatly affected by coarticulation [42], so if we concatenate two speech segments that were not adjacent to each other, there can be spectral or prosodic discontinuities. Spectral discontinuities occur when the formants at the concatenation point do not match. Prosodic discontinuities occur when the pitch at the concatenation point does not match. A listener rates as poor any synthetic speech that contains large discontinuities, even if each segment is very natural.

Thus, when designing a concatenative speech synthesis system we need to address the following issues:

1. What type of speech segment to use? We can use diphones, syllables, phonemes, words, phrases, etc.
2. How to design the acoustic inventory, or set of speech segments, from a set of recordings? This includes excising the speech segments from the set of recordings as well as deciding how many are necessary. This is similar to the training problem in speech recognition.
3. How to select the best string of speech segments from a given library of segments, and given a phonetic string and its prosody? There may be several

strings of speech segments that produce the same phonetic string and prosody. This is similar to the search problem in speech recognition.

4. How to alter the prosody of a speech segment to best match the desired output prosody. This is the topic of Section 16.4.

Generally, these concatenative systems suffer from great variability in quality: often they can offer excellent quality in one sentence and terrible quality in the next one. If enough good units are available, a given test utterance can sound almost as good as a recorded utterance. However, if several discontinuities occur, the synthesized utterance can have very poor quality. While synthesizing arbitrary text is still a challenge with these techniques, for restrictive domains this approach can yield excellent quality. We examine all these issues in the following sections.

We define *unit* as an abstract representation of a speech segment, such as its phonetic label, whereas we use *instance* as a speech segment from an utterance that belongs to the same unit. Thus, a system can keep several instances of a given unit to select among them to better reduce the discontinuities at the boundaries. This abstract representation consists of the unit's phonetic transcription at the minimum, in such a way that the concatenation of a set of units matches the target phonetic string. In addition to the phonetic string, this representation can often include prosodic information.

### 16.3.1. Choice of Unit

A number of units have been used in the field, including context-independent phonemes, diphones, context-dependent phonemes, subphonetic units, syllables, words, and phrases. A compilation of unit types for English is shown in Table 16.4 with their coverage in Figure 16.5.

**Table 16.4** Unit types in English assuming a phone set of 42 phonemes. Longer units produce higher quality at the expense of more storage. The number of units is generally below the absolute maximum in theory, i.e., out of the $42^3 = 74{,}088$ possible triphones, only about 30,000 occur in practice.

| Unit length | Unit type | #Units | Quality |
|---|---|---|---|
| Short | Phoneme | 42 | Low |
| | Diphone | ~1500 | |
| | Triphone | ~30K | |
| | Demisyllable | ~2000 | |
| | Syllable | ~15K | |
| | Word | 100K–1.5M | |
| | Phrase | $\infty$ | High |
| Long | Sentence | $\infty$ | |

The issues in choosing appropriate units for synthesis are similar to those in choosing units for speech recognition (described in Chapter 9):

- The unit should lead to *low concatenation distortion.* A simple way of minimizing this distortion is to have fewer concatenations and thus use long units such as words, phrases or even sentences. But since some concatenations are unavoidable, we also want to use units that naturally lead to "small" discontinuities at the concatenation points. For example, it has been observed that spectral discontinuities at vowels are much more noticeable than at fricatives, or that a discontinuity within a syllable is more perceptually noticeable than a discontinuity across syllable boundaries, and similarly for within-word and across-word discontinuities [55]. Having several instances per unit is an alternative to long units that allows the choice of instances with low concatenation distortion.
- The unit should lead to *low prosodic distortion.* While it is not crucial to have units with the same prosody as the desired target, replacing a unit with a rising pitch with another with a falling pitch may result in an unnatural sentence. Altering the pitch and/or duration of a unit is possible (see Section 16.4) at the expense of additional distortion.
- The unit should be *generalizable*, if unrestricted text-to-speech is required. If we choose words or phrases as our units, we cannot synthesize arbitrary speech from text, because it's almost guaranteed that the text will contain words not in our inventory. As an example, the use of arbitrarily long units in such a way that no concatenation between voiced sounds occurs by cutting at obstruents results in low concatenation distortion but it is shown [47] that over 180,000 such units would be needed to cover 75% of a random corpus. The longer the speech segments are, the more of them we need to be able to synthesize speech from arbitrary text. This generalization property is not needed if closed-domain synthesis is desired.
- The unit should be *trainable.* Our training data should be sufficient to estimate all our units. Since the training data is usually limited, having fewer units leads to better trainability in general. So the use of words, phrases, or sentences may be prohibitive other than for closed-domain synthesis. The other units in Table 16.4 can be considered trainable depending on the limitations on the size of our acoustic inventory.

A practical challenge is how to balance these selection criteria. In this section we compare a number of units and point out their strengths and weaknesses.

#### 16.3.1.1. Context-Independent Phonemes

The most straightforward unit is the phoneme. Having one instance of each phoneme, independent of the neighboring phonetic context, is very generalizable, since it allows us to gen-

erate every word/sentence. It is also very trainable and we could have a system that is very compact. For a language with $N$ phonemes, say $N = 42$ for English, we would need only $N$ unit instances. The problem is that using *context-independent* phones results in many audible discontinuities. Such a system is not intelligible.

### 16.3.1.2. Diphones

A type of subword unit that has been extensively used is the so-called *dyad* or *diphone* [41]. A diphone `s-ih` includes from the middle of the `s` phoneme to the middle of the `ih` phoneme, so diphones are, on the average, one phoneme long. The word `hello /hh ax l ow/` can be mapped into the diphone sequence: `/sil-hh/, /hh-ax/, /ax-l/, /l-ow/, /ow-sil/`. If our language has $N$ phonemes, there are potentially $N^2$ diphones. In practice, many such diphones never occur in the language, so that a smaller number is sufficient. For example, the phonetic alphabet of Chapter 2 has 42 phonemes for English, and only about 1300 diphones are needed. Diphone units were among the first type of unit used in concatenative systems because they yield fairly good quality. While diphones retain the transitional information, there can be large distortions due to the difference in spectra between the stationary parts of two units obtained from different contexts. For example, there is no guarantee that the spectra of `/ax-l/` and `/l-ow/` will match at the junction point, since the instance `/ax-l/` could have been excised from a very different right context than `/ow/` or the instance `/l-ow/` could have been excised from a very different left context than `/ax/`.

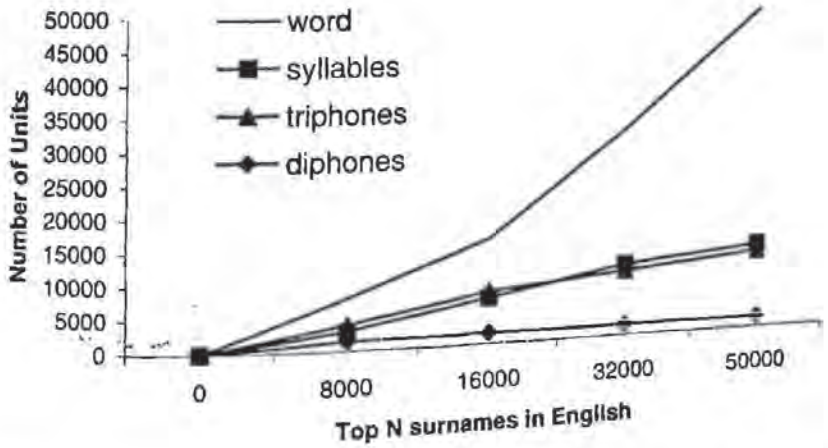



**Figure 16.5** Coverage with different number of units displays the number of units of different types required to generate the top $N$ surnames in the United States [34].

For this reason, many practical diphone systems are not purely diphone based: they do not store transitions between fricatives, or between fricatives and stops, while they store

longer units that have a high level of coarticulation [48]. If only one representative of a dyad is available, there are pitch discontinuities. Prosody modification techniques, such as those described in Section 16.4, can be applied to correct this problem. Otherwise many instances of each diphone are needed for good prosodic coverage. Diphones are trainable, generalizable and offer better quality than context-independent phones.

### 16.3.1.3. Context-Dependent Phoneme

Another subword unit used in the literature [24] is the *context-dependent* phoneme. If the context is limited to the immediate left and right phonemes, the unit is known as *triphone*. As in speech recognition, not all $N^3$ need to be stored, because not all combinations will occur in practice. For English, typically there can be in excess of 25,000 triphones: 12,000 within-word triphones and another 12,000 across-word triphones. Because of the increased number of units, more contextual variations can be accommodated this way. Drawing from experience in speech recognition, we know that many different contexts have a similar effect on the phoneme; thus, several triphones can be clustered together into a smaller number of *generalized triphones*, typically between 500 and 3000. All the clustering procedures described in Chapter 9 can be used here as well. In particular, decision-tree clustered phones have been successfully used. Because a larger number of units can be used, discontinuities can be smaller than in the case of diphones while making the best use of the available data. In addition to only considering the immediate left and right phonetic context, we could also add stress for the current phoneme and its left and right context, word-dependent phones (where phones in particular words are considered distinct context-dependent phones), quinphones (where two immediate left and right phones are used), and different prosodic patterns (pitch ranges and/or durations). As in speech recognition, clustered context-dependent triphones are trainable and generalizable.

Traversing the tree for a given phoneme is equivalent to following the answers for the branching nodes from root to leaves, which determines the clusters for similar context-dependent phones. The decision trees are generated automatically from the analysis database to obtain minimum within-unit distortion (or entropy) for each split. Therefore, one must be able to acquire a large inventory of context-dependent phone HMMs with a decent coverage of the contexts one wishes to model. All the context-dependent phone units can be well replaced by any other units within the same cluster. This method generalizes to contexts not seen in the training data, because the decision tree uses questions involving broad phonetic categories of neighboring contexts, yet provides detailed models for contexts that are represented in the database. Given the assumption that these clustering decision trees should be consistent across different speakers, the use of ample speaker-independent databases instead of limited speaker-dependent databases allows us to model more contexts as well as deeper trees to generate a high-quality TTS voice. These techniques also facilitate the creation of acoustic inventories with a scalable number of units that trade off size with quality. Thus, we can use questions (about the immediate left/right phonetic contexts, stress, pitch, duration,

word, etc.) in the decision-tree clustering methods of Chapter 4 to reduce all the possible combinations to a manageable number.

#### 16.3.1.4. Subphonetic Unit

Subphonetic units, or senones, have also been used with some success [13]. Typically, each phoneme can be divided into three states, which are determined by running a speech recognition system in forced alignment mode. These states can also be context dependent and can also be clustered using decision trees like the context-dependent phonemes. The HMM state has proved to be more effective than the context-dependent phone in speech recognition, also trainable and generalizable, but for synthesis it means having more concatenations and thus possibly more discontinuities. If multiple instances per subphonetic unit are used, higher quality can be obtained.

A *half phone* goes either from the middle of a phone to the boundary between phones or from the boundary between phones to the middle of the phone. This unit offers more flexibility than a phone and a diphone and has been shown useful in systems that use multiple instances of the unit [7].

#### 16.3.1.5. Syllable

It has been observed that discontinuities across syllables stand out more than discontinuities within syllables [55], so syllables are natural units. There are more than 10,000 syllables in English, depending on the exact definition of syllable, so even a context-independent syllable system needs to store at least as many if one instance per syllable is needed for full generalizability. There will still be spectral discontinuities, though hopefully not too noticeable. More than one instance per unit may be needed to account for varying acoustic contexts or varying prosodic patterns, particularly if no waveform modification is to be used.

#### 16.3.1.6. Word and Phrase

The unit can be as large as a word or even a phrase. While using these long units can increase naturalness significantly, generalizability and trainability are poor, so that it is difficult to have all the instances desired to synthesize an output utterance. One advantage of using a word or longer unit over its decomposition in phonemes, as in the above units, is that there is no dependence on a phonetically transcribed dictionary. It is possible that the phoneme string associated to a word by our dictionary is not correct, or not fluent enough, so that using a whole-word model can solve this problem. Of course, the system may have a combination of all units: a set of the most frequent words, sentences, or phrases for best quality some percentage of the time, and some smaller units for full generalizability and trainability.

### 16.3.2. Optimal Unit String: The Decoding Process

The goal of the decoding process is to choose the optimal string of units for a given phonetic string that best matches the desired prosody. Sometimes there is only one possible string, so that this process is trivial, but in general there are several strings of units that result in the same phonetic string yet some of them sound better than others. The goal is to come up with an objective function that approximates this sound quality that allows us to select the best string. The quality of a unit string is typically dominated by spectral and pitch discontinuities at unit boundaries. Discontinuities can occur because of:

1. *Differences in phonetic contexts.* A speech unit was obtained from a different phonetic context than that of the target unit.
2. *Incorrect segmentation.* Such segmentation errors can cause spectral discontinuities even if they had the same phonetic context.
3. *Acoustic variability.* Units can have the same phonetic context and be properly segmented, but variability from one repetition to the next can cause small discontinuities. A unit spoken in fast speech is generally different from another in slow or normal speech. Different recording conditions (amplitude, microphone, sound card) can also cause spectral discontinuities.
4. *Different prosody.* Pitch discontinuity across unit boundaries is also a cause for degradation.

The severity of such discontinuities generally decreases as the number of units increases. More importantly, the prosody of the concatenation has, in general, no resemblance with the prosody specified by the TTS front-end unless we have several instances of each unit, each with a different prosody, or use a prosody modification algorithm (see Section 16.4).

#### 16.3.2.1. Objective Function

Our goal is to come up with a numeric measurement for a concatenation of speech segments that correlates well with how well they sound. To do that we define unit cost and transition cost between two units.

Let $\theta$ be a speech segment with phonetic transcription $p = p(\theta)$. Let $\Theta = \{\theta_1, \theta_2, \cdots, \theta_N\}$ be a concatenation of $N$ speech segments whose combined phonetic transcription is $P = \{p_1, p_2, \cdots, p_N\}$. $P$ is a string of $M$ phonemes, and since each segment has at least one phoneme, it holds that $M \geq N$.

For example, the phonetic string $P =$ "*hh ax l ow*" corresponding to the word *hello* has $M = 4$ phonemes and can be decomposed in $N = 4$ segments $\Theta_1 = \{\theta_1, \theta_2, \theta_3, \theta_4\}$, where $p(\theta_1) = /hh/$, $p(\theta_2) = /ax/$, $p(\theta_3) = /l/$, $p(\theta_4) = /ow/$, each segment being a phoneme. Or it can be decomposed into $N = 2$ segments $\Theta_2 = \{\theta_5, \theta_6\}$, where $p(\theta_5) = /hh\ ax/$,

$p(\theta_6)$ = /l ow/, so that each segment has two phonemes. There are 8 possible such decompositions for this example (in general there are $2^{M-1}$ possible decompositions[1]).

The distortion or cost function between the segment concatenation $\Theta$ and the target $T$ can be expressed as a sum of the corresponding unit costs and transition costs [27, 46] as follows:

$$d(\Theta,T)=\sum_{j=1}^{N} d_u(\theta_j,T)+\sum_{j=1}^{N-1} d_t(\theta_j,\theta_{j+1}) \tag{16.2}$$

where $d_u(\theta_j,T)$ is the *unit cost* of using speech segment $\theta_j$ within target $T$ and $d_t(\theta_j,\theta_{j+1})$ is the transition cost in concatenating speech segments $\theta_j$ and $\theta_{j+1}$. The optimal speech segment sequence of units $\hat{\Theta}$ can be found as the one that minimizes the overall cost

$$\hat{\Theta}=\arg\min_{\Theta} d(\Theta,T) \tag{16.3}$$

over sequences with all possible numbers of units. Transition and unit costs are described in Sections 16.3.2.2 through 16.3.2.5.

Let's analyze the second term in the sum of Eq. (16.2), also shown in Figure 16.6. If all transition costs were identical, the word string with fewest units would have lowest distortion. In practice transition costs are different and, thus, the string with fewest units is not necessarily the best, though there is clearly a positive correlation.

When a large number of speech segments are available, finding the segment sequence with lowest cost is a search problem like those analyzed in Chapter 12. Often a Viterbi algorithm is needed to make this efficient.

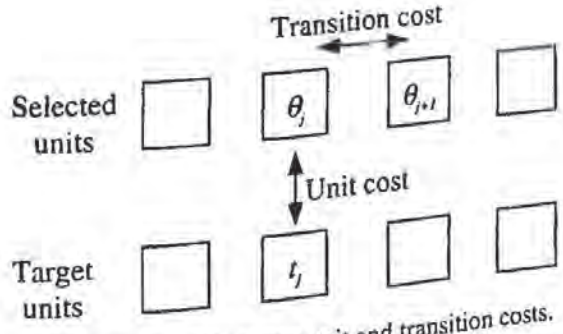



**Figure 16.6** Tradeoff between unit and transition costs.

[1] This assumes that one instance per unit is available. If there are several instances per unit, the number of decompositions grows exponentially.

The art in this procedure is in the exact definition of both transition and unit costs, for which no standard has been defined that works best to date. In Sections 16.3.2.2 and 16.3.2.3 we present an approach for which both transition and unit costs are empirically set after perceptual experiments. Such a system is easy to build, and study of those costs gives insight into the perceptual effects.

Costs obtained using a data-driven criterion are described in Sections 16.3.2.4 and 16.3.2.5. While more complicated than that of empirical costs, this method addresses the shortcomings of the previous method. Finally, we need to estimate some weights to combine the different costs for spectrum and prosody, which can be done empirically or by regression [26].

### 16.3.2.2. Empirical Transition Cost

If spoken in succession, two speech segments have a zero transition cost. But, when they are excised from separate utterances, their concatenation can have varying degrees of naturalness. The transition cost incorporates two types of continuity measures: coarticulatory and prosodic.

An approximation to the prosodic continuity measure is to make it proportional to the absolute difference of the F0 or log F0 at the boundaries, if the boundary is voiced for both units. If we use the prosody modification techniques of Section 16.4, this cost could be set to a small value to reflect the fact that prosody modification is not a perfect process. More sophisticated cost functions can be used to account for prosody mismatches [10].

Regarding the coarticulatory effect, it has been empirically observed that a concatenation within a syllable is more perceptible than when the concatenation is at the syllable boundary. Yi [55] proposed an empirical cost matrix for the concatenation of two speech segments when that concatenation occurs within a syllable (Table 16.5) or at a syllable boundary (Table 16.6). Phonemes are grouped by manner of articulation: vowel/semivowels, fricatives, stops, and nasals. The rows represent the left side of the transition and the columns represent the right side, and NA represents a case that does not occur. These costs reflect perceptual ratings by human listeners to unit concatenations between different phonemes. Values of 10, 2000, 5000, 7500, and 10,000 were used to indicate different degrees of goodness from very good to very bad concatenations.

**Table 16.5** Cost matrix for intrasyllable concatenations (after Yi [55]). The rows represent the left side of the transition and the columns represent the right side, and NA represents a case that does not occur.

| | vowel | semivowel | nasal | obstruent | /h/ |
|---|---|---|---|---|---|
| vowel | 10,000 | 10,000 | 7500 | 10 | NA |
| semivowel | 10,000 | 7500 | 7500 | 10 | NA |
| nasal | 5000 | 10 | NA | 10 | NA |
| /h/ | 5000 | NA | NA | NA | NA |
| obstruent | 10 | 10 | 10 | 10,000 | NA |

Table 16.6 Cost matrix for intersyllable concatenations (after Yi [55]). The rows represent the left side of the transition and the columns represent the right side, and NA represents a case that does not occur.

| | vowel | semivowel | nasal | obstruent | /h/ | silence |
|---|---|---|---|---|---|---|
| vowel | NA | 7500 | 5000 | 10 | 5000 | 10 |
| semivowel | 7500 | 7500 | 2000 | 10 | 10 | 10 |
| nasal | 2000 | 10 | 10 | 10 | 10 | 10 |
| obstruent | 10 | 10 | 10 | 5000 | 10 | 10 |
| /h/ | NA | NA | NA | NA | NA | NA |
| silence | 10 | 10 | 10 | 10 | 10 | 10 |

### 16.3.2.3. Empirical Unit Cost

The unit cost is generally a combination of the coarticulation cost and the prosodic cost. Prosodic mismatches can be made proportional to the F0 difference between the candidate unit and the target unit or set to a fixed low value if the prosody modification techniques of Section 16.4 are used.

A way of determining the cost associated with replacing a phonetic context with another was proposed by Yi [55], who empirically set cost matrices for phone classes by listening to concatenations where such contexts were replaced. These ad hoc values also bring some sense of where the coarticulation problems are. Replacing a vowel or semivowel by another with a context that has a different place of articulation or nasalization results in audible discontinuities. The rows represent the context class of the target phoneme and the columns represent the context class of the proposed unit. Table 16.7, Table 16.8, Table 16.9, Table 16.10, Table 16.11, and Table 16.12 include an empirical set of costs for such mismatches between the target's context and a candidate unit's context for the case of vowel/semivowels, fricatives, stops, and nasals. These costs reflect human listeners' perceptual ratings of speech units with an incorrect phonetic context. Values of 10, 100, 500, and 1000 were used to indicate very good, good, bad, and very bad units. These values are chosen to match the values for transition costs of Section 16.3.2.2.

Table 16.7 Unit coarticulation cost matrix (after Yi [55]) for left and right context replacements for vowels and semivowels.

| | labial | alv/den/pal | velar | m | n | ng | front | back | none |
|---|---|---|---|---|---|---|---|---|---|
| labial | 10 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 |
| alv/den/pal | 1000 | 10 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 |
| velar | 1000 | 1000 | 10 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 |
| m | 1000 | 1000 | 1000 | 10 | 1000 | 1000 | 1000 | 1000 | 1000 |
| n | 1000 | 1000 | 1000 | 1000 | 10 | 1000 | 1000 | 1000 | 1000 |
| ng | 1000 | 1000 | 1000 | 1000 | 1000 | 10 | 1000 | 1000 | 1000 |
| front | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 10 | 1000 | 1000 |
| back | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 10 | 1000 |
| none | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 1000 | 10 |

**Table 16.8** Unit coarticulation cost matrix (after Yi [55]) for left and right context replacements for fricatives.

| | retroflex | round | sonorant | other |
|---|---|---|---|---|
| retroflex | 10 | 100 | 100 | 100 |
| round | 100 | 10 | 100 | 100 |
| sonorant | 100 | 100 | 10 | 100 |
| other | 100 | 100 | 100 | 10 |

**Table 16.9** Unit coarticulation cost matrix (after Yi [55]) for left context replacements for stops.

| | front | back | retroflex | round | other |
|---|---|---|---|---|---|
| front | 10 | 10 | 10 | 10 | 10 |
| back | 10 | 10 | 10 | 10 | 10 |
| retroflex | 10 | 10 | 10 | 10 | 10 |
| round | 10 | 10 | 10 | 10 | 10 |
| other | 500 | 500 | 500 | 500 | 10 |

**Table 16.10** Unit coarticulation cost matrix (after Yi [55]) for right context replacements for stops.

| | front | back | retroflex | round | schwa | other |
|---|---|---|---|---|---|---|
| front | 10 | 100 | 100 | 100 | 500 | 100 |
| back | 100 | 10 | 100 | 100 | 500 | 100 |
| retroflex | 100 | 100 | 10 | 100 | 500 | 100 |
| round | 100 | 100 | 100 | 10 | 500 | 100 |
| schwa | 500 | 500 | 500 | 500 | 10 | 500 |
| other | 100 | 100 | 100 | 100 | 500 | 10 |

**Table 16.11** Unit coarticulation cost matrix (after Yi [55]) for left context replacements for nasals.

| | obstruent | sonorant |
|---|---|---|
| obstruent | 10 | 1000 |
| sonorant | 1000 | 10 |

Table 16.12 Unit coarticulation cost matrix (after Yi [55]) for right context replacements for nasals.

| | voiced | unvoiced | sonorant |
|---|---|---|---|
| voiced | 10 | 100 | 1000 |
| unvoiced | 100 | 10 | 1000 |
| sonorant | 1000 | 1000 | 10 |

### 16.3.2.4. Data-Driven Transition Cost

The empirical transition costs of Section 16.3.2.2 do not necessarily mean that a spectral discontinuity will take place, only that one is likely, and that if it occurs within a syllable it will have a larger perceptual effect than if it occurs across syllable boundaries. While that method can result in a good system, the cost is done independently of whether there is a true spectral discontinuity or not. Thus, it has been also proposed to use a measurement of the spectral discontinuity directly. This is often estimated as:

$$d_t(\theta_i,\theta_j)=\left|\mathbf{x}_i(l(\theta_i)-1)-\mathbf{x}_j(0)\right|^2 \tag{16.4}$$

the magnitude squared of the difference between the cepstrum at the last frame of $\theta_i$ and the first frame of $\theta_j$. The quantity $l(\theta_i)$ denotes the number of frames of speech segment $\theta_i$, and $\mathbf{x}_i(k)$ the cepstrum of segment $\theta_i$ at frame $k$.

This technique can effectively measure a spectral discontinuity in a region with slowly varying spectrum, but it can fail when one of the segments is a nasal, for example, for which a sharp spectral transition is expected and desired. A better way of measuring this discontinuity is shown in Figure 16.7, in which we measure the cepstral distance in an overlap region:[2] the last frame of segment 1 and the first frame before the beginning of segment 2:

$$d_t(\theta_i,\theta_j)=\left|\mathbf{x}_i(l(\theta_i)-1)-\mathbf{x}_j(-1)\right|^2 \tag{16.5}$$

When many speech segments are considered, a large number of cepstral distances need to be computed, which in turn may result in a slow process. To speed it up an approximation can be made where all possible cepstral vectors at the boundaries are vector quantized first, so that the distances between all codebook entries can be precomputed and stored in a table.

[2] This means extra frames need to be stored.

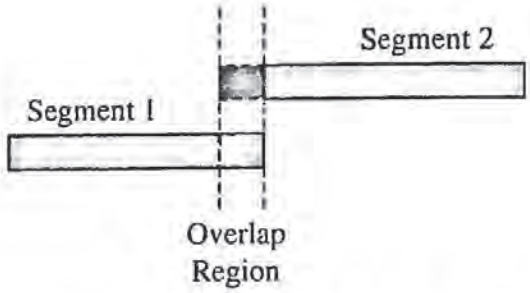



**Figure 16.7** Measurement of the spectral discontinuity in the overlap region. The dark gray area is the speech region that precedes segment 2 and does not form part of segment 2. This area should match the last part of the segment 1.

A spectral discontinuity across, say, fricatives is perceptually not as important as if it happens across vowels [48]. For this reason, the cepstral distance described above does not correlate well with perceptual distances. To solve this problem, it is possible to combine both methods, for example by weighting the spectral/cepstral distance by different values.

Even if no spectral discontinuity is present, a phase discontinuity may take place. The pitch periodicity may be lost at the boundary. This can be generally solved by fine adjustment of the boundary using a correlation approach as described in Section 16.4. You need to keep in mind that such methods are not perfect.

### 16.3.2.5. Data-Driven Unit Cost

Spectral discontinuities across concatenations are often the result of using a speech segment with a different phonetic context than the target. One possibility is, then, to consider only speech segments where the phonetic contexts to the left and right match exactly. For example, if we use a phoneme as the basic speech segment, a perfect match would require on the order of at least 25,000 different segments. In this case, the coarticulation unit cost is zero if the target and candidate segment have the same phonetic context and infinite otherwise. When longer segments are desired, this number explodes exponentially. The problem with this approach is that it severely reduces the number of potential speech segments that can be used.

Generalized triphones, as described in Section 16.3.1.3, are used in [24]. In this approach, if the speech segments have the same generalized triphone contexts as the target utterance, the unit cost is zero, otherwise the cost is infinite. The technique allows us to use many more possible speech segments than the case above, yet it eliminates those speech segments that presumably have context mismatches that in turn lead to unnatural concatenations. When using a large training database, it was found that bringing the number of triphones from 25,000 down to about 2000 did not adversely impact the quality, whereas some degradation was perceived when using only 500 phoneme-length segments. Thus, this technique allows us to reduce the size of the speech segment inventory without severely degrading the voice quality.

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

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]

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.

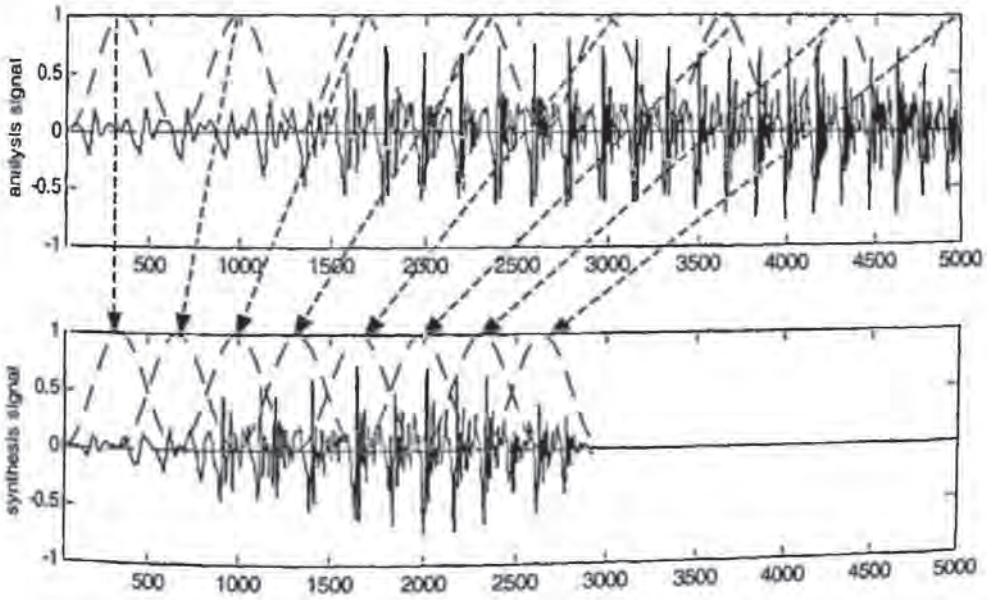



**Figure 16.8** Overlap-and-add (OLA) method for time compression. Hanning windows of length $2N$, $N = 330$, are used to multiply the analysis signal, and resulting windowed signals are added. The analysis windows, spaced $2N$ samples, and the analysis signal $x[n]$ are shown on the top. The synthesis 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.

## 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 $x_i[n]$

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

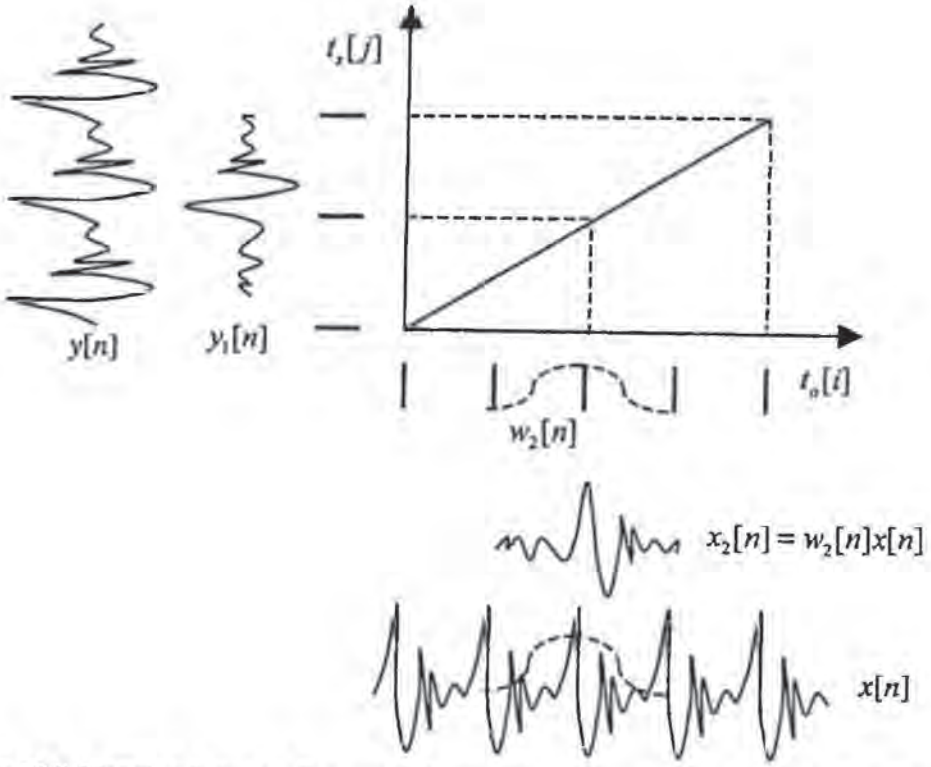



**Figure 16.9** Mapping between five analysis epochs $t_a[i]$ and three synthesis epochs $t_s[j]$. Duration has been shortened by 40% and pitch period increased by 60%. Pitch cycle $x_2[n]$ is the product of the analysis window $w_2[n]$, in dotted line, with the analysis signal $x[n]$, which is aligned 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.

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 $x_i[n]$ 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.

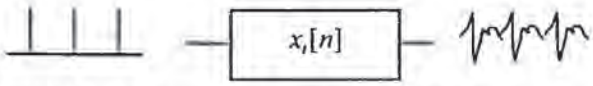



**Figure 16.10** PSOLA technique as an impulse train driving a time-varying filter.

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.

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-

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

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

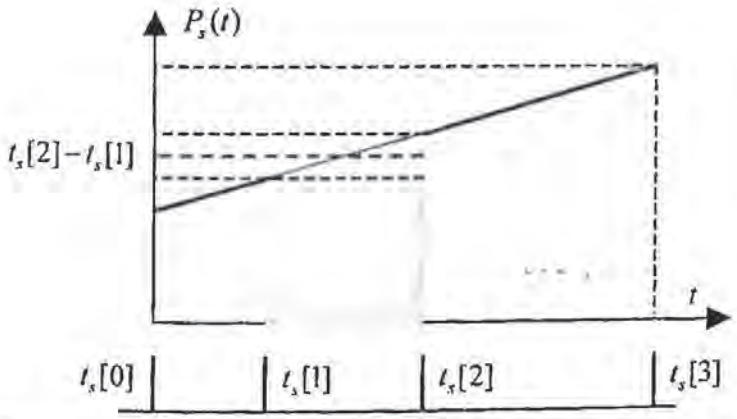



**Figure 16.11** The desired pitch period $P_s(t)$ is a linearly increasing function of time such that the pitch period is doubled by the end of the segment. The four synthesis epochs $t_s[j]$ are computed to satisfy Eq. (16.15). In particular, $t_s[2]$ is computed such that $t_s[2] - t_s[1]$ equals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

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 $\delta_j$ 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 $t_s[j+1]$.

## 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 $t$.

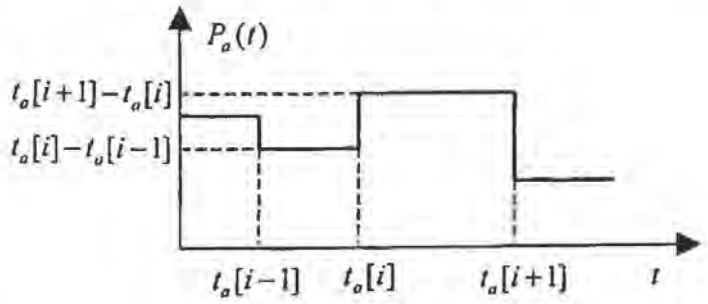



**Figure 16.12** Pitch period of the analysis waveform as a function of time. It is a piecewise constant function of time.

### 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 $t$.

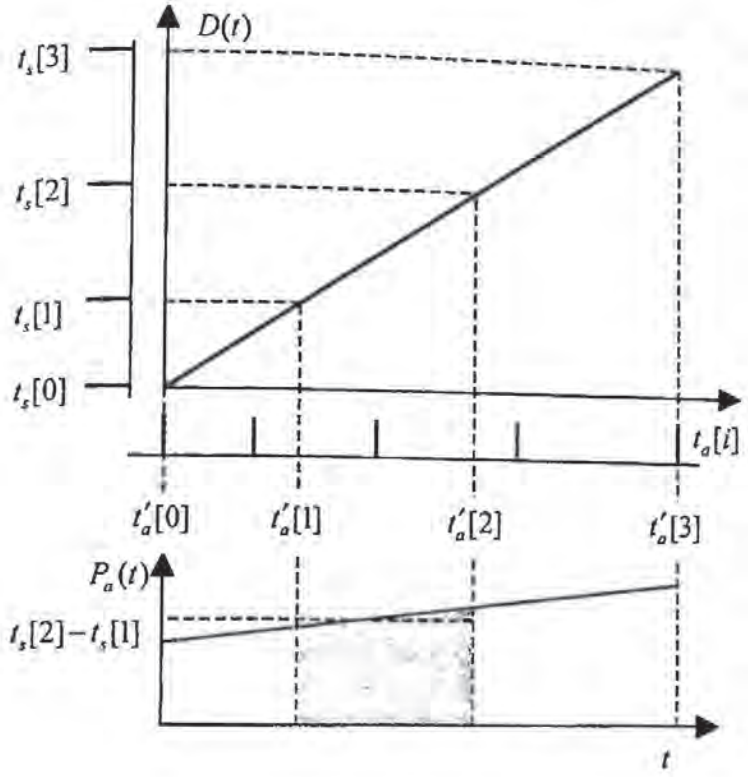



**Figure 16.13** Time-scale modification of speech. The five analysis epochs $t_a[i]$ are shown in the $x$-axis and the four synthesis epochs $t_s[i]$ in the ordinate. Duration is shortened by 25% while maintaining the same pitch period. The corresponding virtual analysis epochs $t'_a[i]$ are obtained 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

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

where $\gamma_j$ 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 $\gamma_j$ 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 $\alpha$ 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.

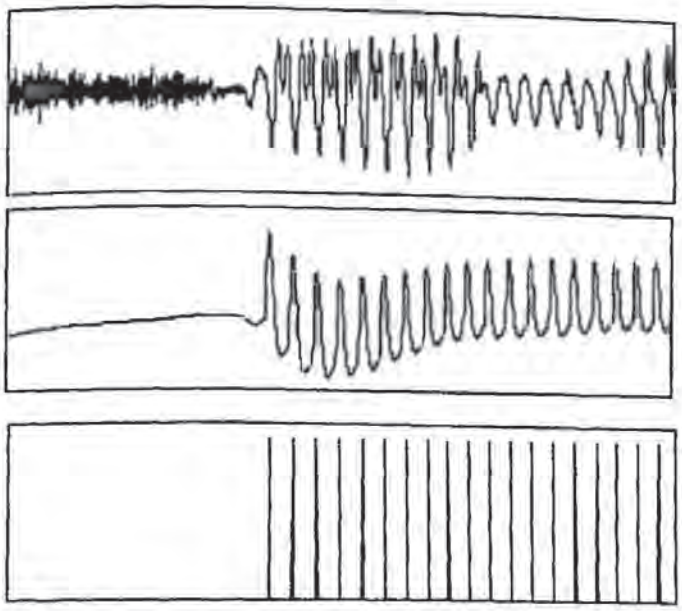


**Figure 16.14** Speech signal, laryngograph signal, and its corresponding epochs.

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.

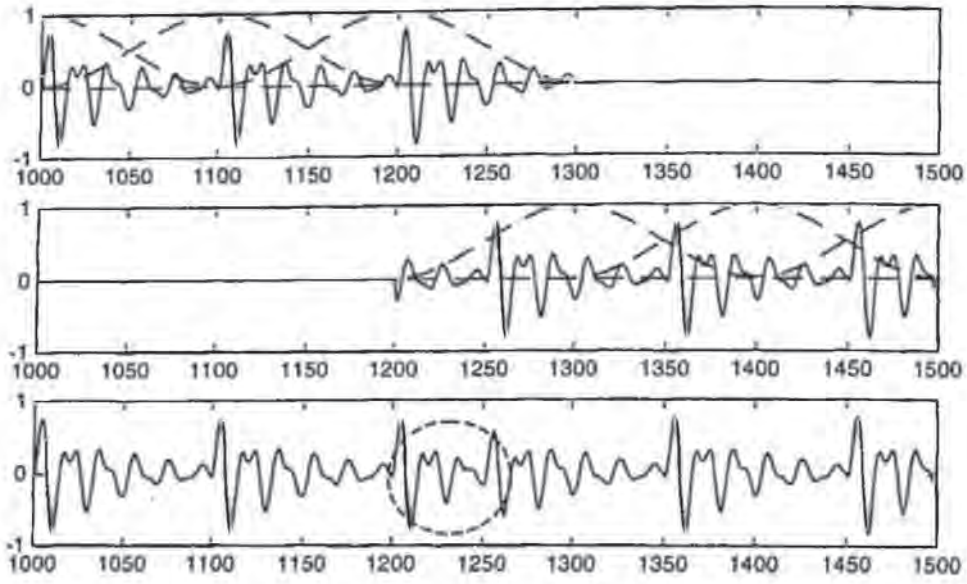



**Figure 16.15** Phase mismatches in unit concatenation. Waveforms are identical, but windows are not centered on the same relative positions within periods.

- *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).

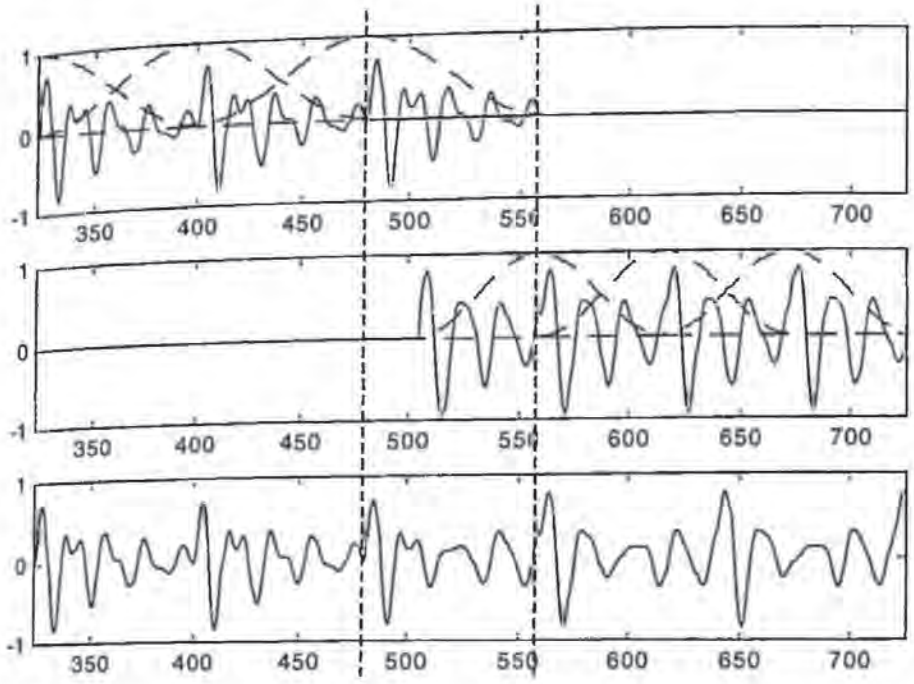



**Figure 16.16** Pitch mismatches in unit concatenation. Two synthetic vowels were generated with a pitch of 138 Hz (top) and 197 Hz (middle) and exactly the same transfer function. There is no pitch tracking error, and windows are positioned coherently (no phase mismatch). The pitch of the second wave 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.

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

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:

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

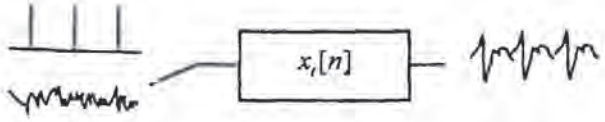



Figure 16.17 Speech synthesis model with white noise or an impulse train driving a time-varying filter.

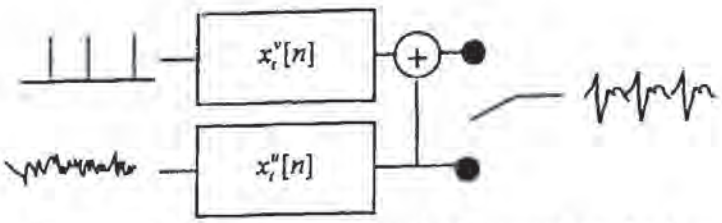



**Figure 16.18** Mixed excitation model for speech synthesis.

where $A_k(t)$ and $\phi_k(t)$ are the amplitude and phase at time $t$ of the $k$th 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.

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.

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.

### 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 |

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 |

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 |

### 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 |

**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 |

### 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

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.

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.

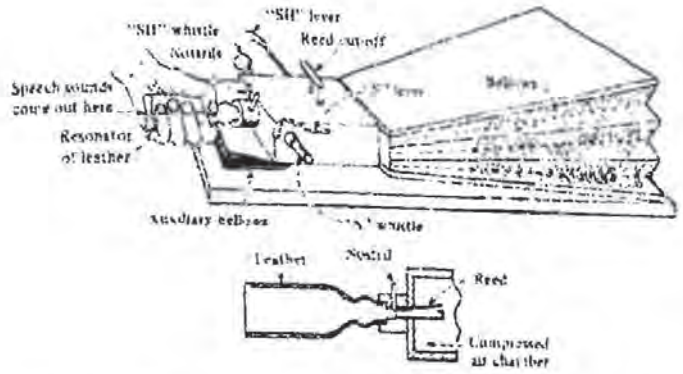



**Figure 16.19** Wheatstone's reconstruction of von Kempelen's speaking machine [14] (after Flanagan [17]).

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.

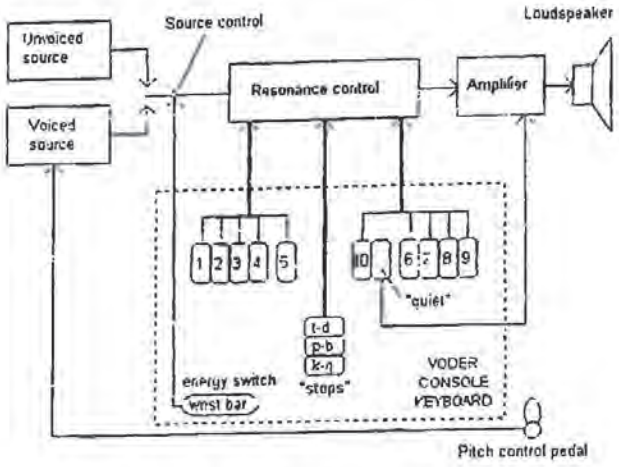



**Figure 16.21** Block diagram of the Voder by Homer Dudley, 1939 (after Flanagan [17]).

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

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.

[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.

[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.

[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.

# 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.

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 **X**, the objective of the system is to arrive at actions **A** (including dialog messages and necessary operations) so that the cost of choosing **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 **F** denotes semantic interpretation of **X** and $S_n$, the discourse semantics for the $n^{\text{th}}$ 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 **X** into a collection of semantic objects **F**. This component is often further decomposed into *speech recognition* module (converting speech signal **X** into textual sen-

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.

### 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

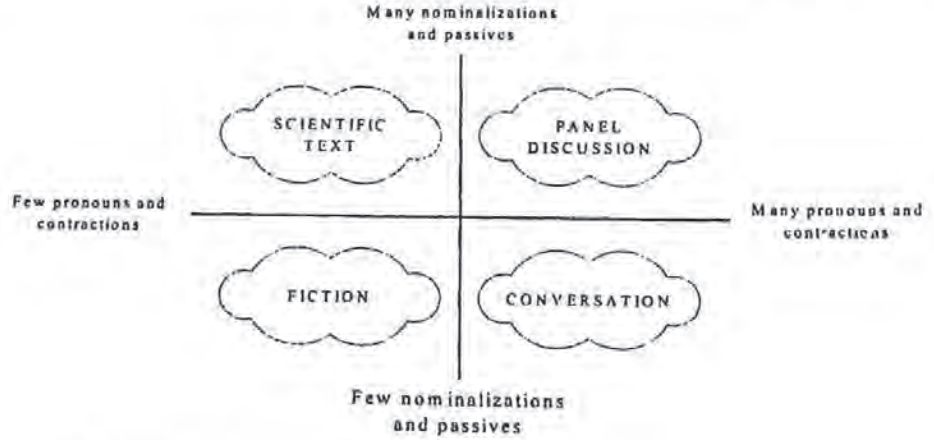



**Figure 17.1** Dimensions of written vs. spoken language variation.

[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.*

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.

### 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.

## 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.

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.

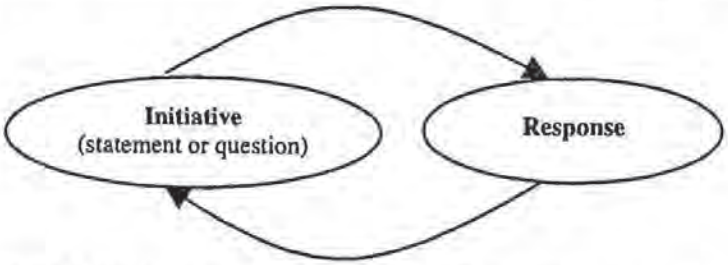



**Figure 17.2** The fundamental structure of dialog: initiative and response.

## 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

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.

**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:

- **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! |

**Table 17.3** Backward looking tags.

| Backward Looking Tags | Example |
|---|---|
| accept | (Will you come?)<br>Yes. [and/or, I'll be there at 10.] |
| accept-part | (Will you come with your wife?)<br>I'll come, she may be busy. |
| reject | (Will you come?)<br>No. |
| reject-part | (Want fries and a shake with that burger?)<br>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?)<br>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.

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

*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-

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.

```
[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-

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,

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.

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.

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*.

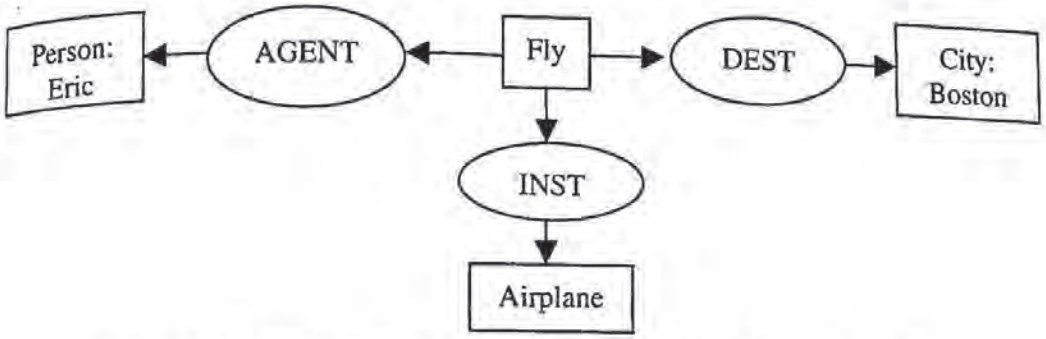



**Figure 17.11** CG display form for *Eric is flying to Boston by airplane* [53].

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.

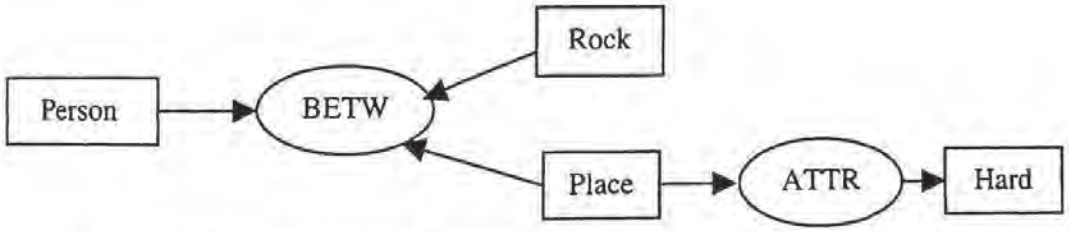



**Figure 17.12** CG display form for *a person is between a rock and a hard place* [53].

```
[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

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
```

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.

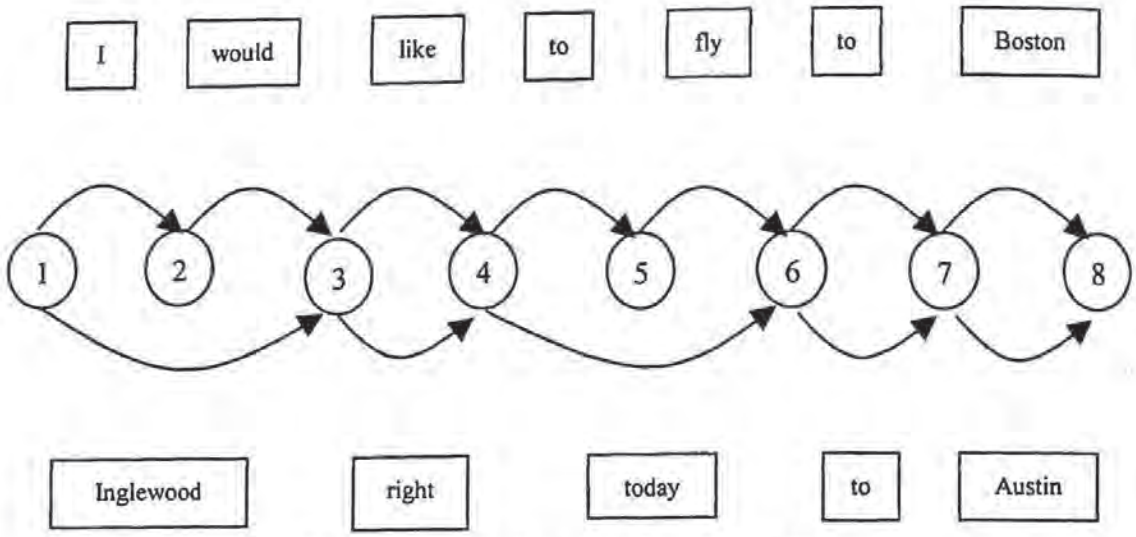



**Figure 17.14** Word graph for hypotheses [61].

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.

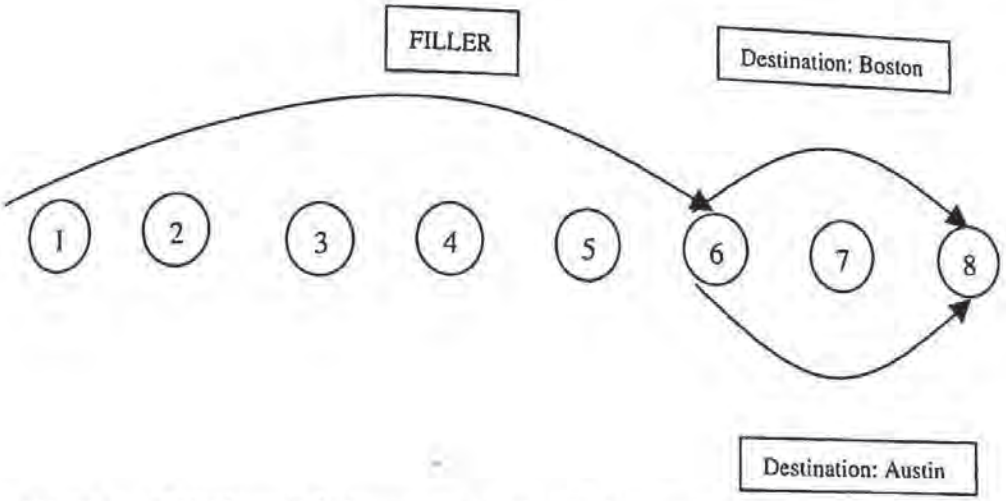



**Figure 17.15** Word graph for hypotheses if only the *Destination* grammar is active [61].

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.*"

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

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).

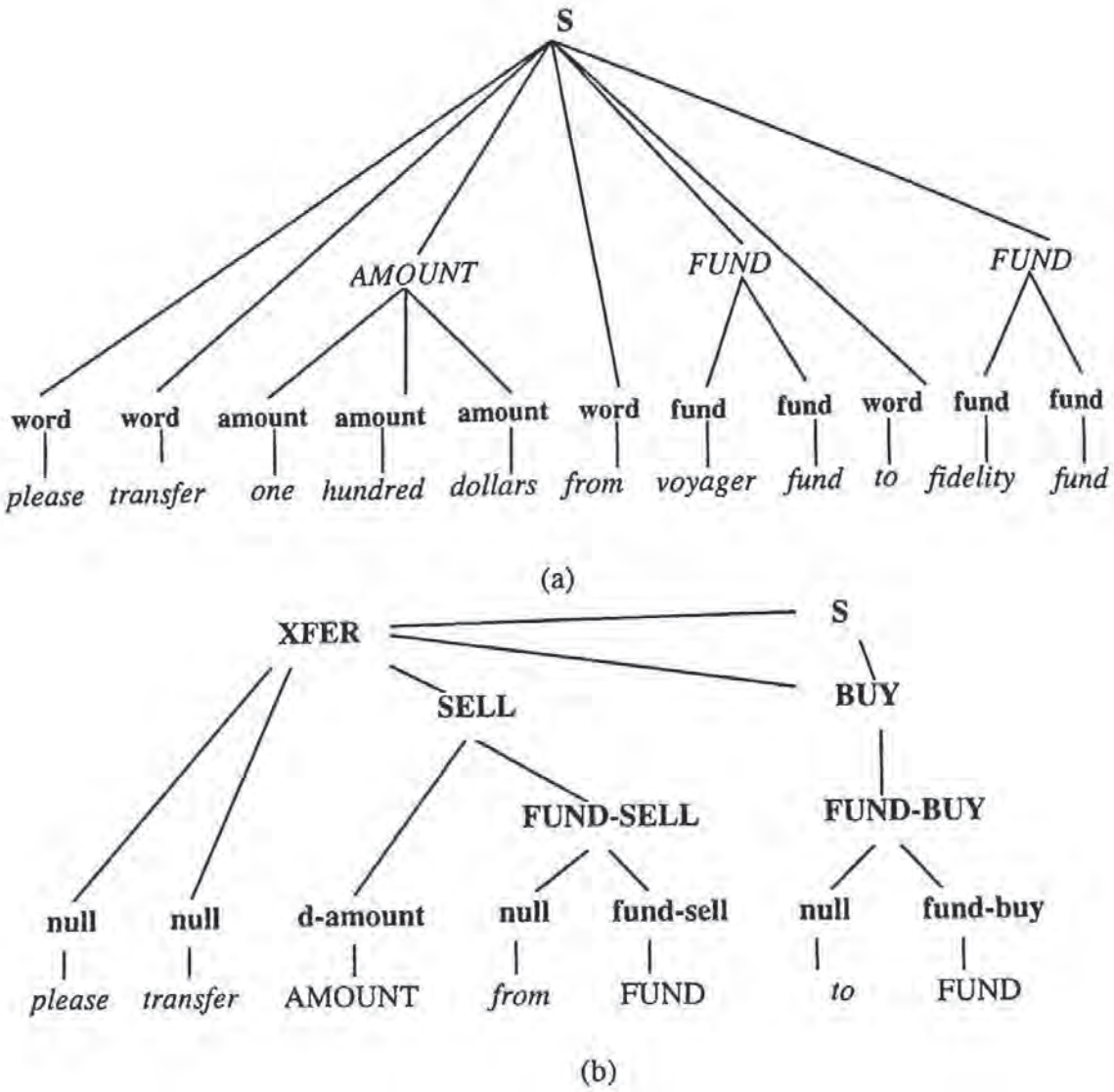



**Figure 17.16** An example class tree in IBM's statistical class parser. (a) The sentence is classified into semantic classes. (b) The classed sentence is parsed into the semantic tree based on CART [15].

```
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].

## 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.

### 17.5.1. Resolution of Relative Expression

There are two types of relative expression. The first type is the *reference*, relating linguistic expressions to real-world entities. This may involve disambiguation, by inference or direct user query. When a user says, "*Give me Eric's phone number,*" many people with first name *Eric* may exist in the database. The second type of relative expression is the *co-reference*. Co-reference occurs when different names or referring expressions are used to signify the same real-world entity. For example, in the sentence "*Nelson Mandela* has a long history of leadership within the African National Congress, but *he* is aging and nobody was surprised yesterday when *Mandela* announced his successor" the terms *Nelson Mandela* and *Mandela* refer to the same person.

In linguistics, there are three different types of co-references. The example above is an *ellipsis*, where the omitted word(s) can be understood from the context. The other type is *deixis*. A deixis refers to the use of a word such as that, now, tomorrow, or here, whose full meaning depends on the extralinguistic context in which it is used. Location deictic co-references are very common for multimodal applications where pointing devices (modalities) like pens can be used to indicate the real locations. The most common type of co-reference is *anaphora*, which is a special type of co-reference, where a word or phrase has an indirect, dependent meaning, standing for another word or concept previously introduced. The pronoun *he* in the sentence above is an anaphor referring to *Nelson Mandela* too.

Time deictic co-references like *tomorrow*, *next week, the week of 2/ 7/ 2000*, etc., are among the easiest category for resolution (requiring only simple domain knowledge). The resolution of other relative expressions usually requires deep natural language processing. We focus our discussion on anaphora resolution, since it represents the most challenge one among others and approaches of solving this problem are typical of the kind of methods appropriate for resolving a variety of other relative expressions.

#### 17.5.1.1. Priority Entity Memory

We introduce a simple resolution method [60] that is based on semantic class type abstraction and priority entity memory. This method is straightforward and is very powerful to handle most cases even without complex natural language processing.

Whenever a conversion of a relative expression occurs, the consequent entity is added to the entity memory. The entity memory consists of *turn* and *discourse* memories. Either type of memory consists of a number of priority queues that are delineated by entity types. An entity can only be remembered into the queue of compatible types (e.g., through inheritance). When referred to, the memory item increases its priority in the queue. This treatment resembles the *cache language* model described in Chapter 11.

The turn memory is a cache for holding entities in each turn. There are two types of turn memories. The *explicit* memory holds the entities that are resolved directly from semantic objects. In contrast, the *implicit* memory is for entities that are deduced from relative

expressions. In accessing the memory, the explicit turn memory takes precedent over the discourse memory, which in turn has a higher priority than the implicit. At the end of the system's turn, all the turn memory items are moved and sorted into the discourse memory.

The distinctions between the three kinds of memories and the rules to operate them are designed as a simple mechanism for most common but not all possible scenarios. It is worth noting that the design has a bias toward *direct* and *backward* reference. For example, in the expression "*Forward this mail to John, his manager, and his assistant,*" the second *his* will be evaluated as referring to *John*, not to *his manager*. The implicit memory, however, provides a back-off for expressions like "*Send email to John, his manager, and her assistant*" in which the pronoun *her* should be taken as indicating John's manager is a female and resolved accordingly. However, since we store only the entities and not the semantic objects into the memory, the mechanism is not suitable for forward or pleonastic references, as in the examples like "*Since his promotion last May, John has been working very hard*" or "*It being so nice, John moved the meeting outside.*" Fortunately, these natural language phenomena are rare in a spoken dialog environment.

It is sensible to confirm[3] the resolved entities with users due to possible resolution errors. In cases where many entities in the entity memory can be matched with a semantic object, a decision of not performing any resolution and directly inquiring the user for disambiguation may be a better solution. In general, name references can be resolved by a sequence of simple rules. In the example of "*Give me Eric's phone number*" the SLU system may just generate the query message "*What is Eric's last name*?" when many people in the entity memory have the same name *Eric*.

### 17.5.1.2. Resolution by NLP

Extensive understanding is crucial for perfect resolution for relative expressions (in particular, anaphora). Though morphology, lexical semantics, and syntax can be helpful for disambiguation, ultimately it is a problem of inference using real-world knowledge and dialog state or context. In a discourse model of focus, it is assumed that speakers usually center their attention on a single main topic called the *focus*. Some utterances introduce or reintroduce a focus; others elaborate on it. Focus elements typically change (by being suspended, interrupted, resumed, etc.) over the course of a dialog. Once a focus element has been introduced, anaphora is usually used to represent it, making dialog more efficient.

Anaphora resolution specifies the referent of a pronoun or other anaphoric expression. This association should be supported by inference about properties and probabilities in the real world. Anaphora resolution can be done with a simple *entity focus* principle. For example, in the very common *schedule a meeting* type of dialog application, an exchange such as that shown in Figure 17.18 is centered on the initial focus element—the proposed meeting—and anaphora are likely to relate to that central topic, at least early on in the exchange. The

[3] One might decide which confirmation strategies (explicit or implicit confirmation) to use based on the confidence of the resolutions. The details of confirmation strategies are described in Section 17.6.

(1) I'd like to schedule a [meeting]$_i$ with [Christoph]$_j$.
(2) [It]$_i$ can be anytime after 4.
(3) Tell [him]$_j$ [he]$_j$ can [grab a cab over here]$_k$.
(4) [That]$_k$ should be only if he's running late.

**Figure 17.18** A *schedule a meeting* dialog example showing different anaphora usage.

subscript indicates the co-reference to the same entity. The focus is the *meeting* proposed in (1). The pronoun *it* in (2), by the very simple mechanism discussed here, can be interpreted as referring to the meeting. Some grammatical knowledge and the semantic class type should help the system to resolve *him* in (3) as *Jim* rather than the *meeting*. In sentence (3) the focus has shifted to the action of *taking a cab*, to which *that* refers in sentence (4). The locative *here* in (3) must also be resolved to the speaker's location.

Most formal models of anaphora resolution originated from research into discourse and human-human dialog. They tend to be overpowered, in making elaborate provision for greater topic and reference variation than exists in typical computer speech dialog applications of the present time. On the other hand, while they can provide resolution for some complicated situations, they tend to be underpowered, in failing to deal robustly with the realities of imperfect speech recognition and parsing.

Some of the work on anaphora resolution in dialog relies on elaborate focus-tracking mechanisms [47]. These tend to be somewhat circular in nature, in that anaphoric reference resolution is required for the focus-tracking algorithms to operate, while the anaphoric resolution itself relies on the currently identified focus structure of the dialog or discourse. Rather than elaborate on these possibilities, we instead present a number of relatively straightforward heuristics for anaphora resolution, some of which have been developed based on textual studies, but which may be relevant to increasingly complex human-computer dialog in the future. The discussion here is limited to the resolution of intersentential and intrasentential pronominal anaphora. Full noun-phrase anaphora, where one synonymous noun phrase is co-referent with another, requires even more powerful grammatical and semantic resources.

Syntactic conditions can be tested when a parse tree showing syntactic constituency is available. The most obvious syntactic filter for disallowing co-reference is simple grammatical feature agreement. For example, the following proposed co-indexed relation is not semantically possible in ordinary discourse, and the restriction is explicitly provided through the lexical morphology and syntax of the language:

The [girl]$_i$ thought [he]$_i$ was frightening.

Though the theoretical details can be complex [37], the basic intuition of syntax-based anaphoric resolution is that nonreflexive pronouns that are syntactically too close to a candidate co-referential NP (antecedent) are disfavored. For example, in a sentence such as:

[Bill's]$_i$ photo of [him]$_i$ is offensive.

the coindexing of *Bill* with *him* is disallowed. By disallowed, we mean that your innate sense of proper English grammar and interpretation will balk at the proposed relation. The language provides a mechanism to override some proximity restrictions, as in the following repaired version:

```
[Bill's]_i photo of [himself]_i is offensive.
```

So, when is a pronoun *too close* to a possible antecedent? The most important syntactic concepts for determining anaphoric relations rely on structural attributes of parse trees. In fact, treatment of this problem represents a very large and highly argumentative subfield within theoretical linguistics. Nevertheless, any treatment of anaphora resolution on purely syntactic grounds is very likely to end with a list of conditions that can mostly be subsumed under some form of *x-bar* theory [25], as it is called in the theoretical linguistics.

### 17.5.2. Automatic Inference and Inconsistency Detection

Automatic inference can be carried out through the same framework of priority entity memory described in Section 17.5.1.1. During semantic evaluation, a partially filled semantic object is first compared with the entities in the memory based on the type compatibility. If a candidate is found, the discourse analysis module then computes a goodness-of-fit score by consulting the knowledge base and considering the position of the entity in the memory list. The semantic object is converted immediately to the entity from the memory if the score exceeds the threshold. In the process, all the actions implied by the entities are carried out following the order in which the corresponding semantic objects are converted.

In general, automatic inference can be implemented as description procedures attached to semantic slots as described in Section 17.3.1. In the example of an airline reservation system, a procedure or rule can be attached to automatically infer the destination city for the returning flight. The other powerful strategy for automatic inference is *slot inheritance*. When changing dialog turn for different semantic objects under the same service, the system may allow such slot inheritance to free users from repeating the same attributes. For example, after a user asks "*What is Peter Hon's office number*?" he may abbreviate his next query to "*How about Derek Acero's*?" Slot inheritance will allow the second semantic object regarding *Derek Acero* to inherit the *office number* slot even though it is not explicitly specified.

*Inconsistency checking* is crucial to initiate necessary events for *dialog repair*. A dialog may be diverted away from the ideal flow for various reasons (e.g., misrecognition, out-of-domain reference, conflicting information), many of which require domain- and application-specific knowledge to guide the dialog back to the desired course. This process is called dialog repair. Similar to automatic inference, inconsistency checking can be implemented as description procedures attached to semantic slots. In addition, inconsistency checking can also be triggered when semantic binding for a partially filled semantic object fails (e.g., indicated by a failed database lookup). The discourse analysis module is responsible only for

sending the dialog repair events to the dialog manager, and it leaves the realization of the repair strategy to the corresponding event handler in the dialog manager.

For example, consider a query: "*Find me the cheapest flight from Seattle to Memphis on Sunday.*" The semantic binding fails because there is actually no flight available on Sunday from Seattle to Memphis based on the flight database. Thus, the discourse manager passes such event to the dialog manager, and the dialog manager will generate an appropriate message to let the users be aware of this fact.

## 17.6. DIALOG MANAGEMENT

For most applications, it is highly unlikely that a user can access or retrieve the desired information with just a single query. The query might be incomplete, imprecise, and sometimed inconsistent with respect to the discourse history. Even if the query is unambiguous, the speech recognition and sentence interpretation modules in a SLU system may make mistakes. Thus the SLU system needs to provide an interactive mechanism to perform clarification, completion, confirmation, and negotiation dialogs with users. By default, the objective of such a dialog is to help users accomplish the required tasks more efficiently. Being user-friendly is also one of the major objectives for dialog systems as discussed in Chapter 18. Since the goal of a SLU system is to provide a natural conversation interface for users, the ultimate SLU system should act like a real human, yet still possessing perfect memory and superfast computation. Based on these criteria, it is not hard to see why mix-initiative systems are preferred over system-initiative systems.

The dialog manager controls the interactive strategy and flow once the semantic meaning of the query is extracted and stored in the system's representation (discourse trees). The architecture of SLU dialog systems resembles the one used in event-driven GUI systems. In the same way that GUI events are assigned to graphical objects, the dialog events are assigned to semantic objects that encapsulate the knowledge for handling events under various discourse contexts. As mentioned in Section 17.5, the discourse tree with domain entity binding is passed along with necessary dialog events generated from the discourse analysis module to the dialog manager. The dialog manager acts as an intelligent domain knowledge handler that uses the semantic meaning of the query to check against domain-specific knowledge (including domain database and application logic) and generates the desired answer for the query or produces other necessary dialog strategy.

In this sense, the dialog manager functions as a GUI application that contains an event handler. The event handler handles dialog events passed from the discourse analysis module and generates appropriate responses to engage users to solve the problems. In addition, the dialog manager needs to implement the application logic to generate appropriate actions (e.g., make real airline and hotel reservation). In this section we discuss two modeling techniques for implementing application logic, and different dialog behaviors related to event handling.

### 17.6.1. Dialog Grammars

*Dialog grammars* use constrained, well-understood formalisms such as *finite state* machines to express sequencing regularities in dialogs, termed *adjacency pairs*. The rules state sequential and hierarchical constraints on acceptable dialogs, just as syntactic grammar rules state constraints on grammatically acceptable strings. For example, an answer or a request for clarification is likely to follow a question, just as a finite state grammar might provide for a noun or an adjective, but not a verb, to follow a determiner such as *the*. In most dialog grammar systems, dialog-act types (*explain*, *complain*, *request*, etc. cf. Section 17.2.2) are categorized ,and the categories are used as terminals in the dialog grammar. This approach has the advantage that the formalism is simple and tractable. At every stage of processing the system has a basis for setting expectations, which may correspond to activating state-dependent language models, and for setting thresholds for rejection and requests for clarification.

In its essence, the dialog grammar model is exemplified by a rigid flowchart diagramming system control of the type and sequence of interaction. Figure 17.19 shows a finite state dialog grammar for an airline reservation SLU system. In this simple example, dialog-act categorization is omitted, and the interactions are controlled based on bare information items. This grammar makes simple claims: the interaction is basically question-answer; the topic queries are answered on-topic if possible, and presumably with a confirmation statement to catch the existence of a problem.

This system is easily programmed. The challenge lies in providing tools to application authors to ease the tedium and minimize the errors in the construction of grammars, and to allow for more flexibility and spontaneous deviations from the expected transitions in the grammar. Such deviations may be important for novice users, who may more naturally tend to give their information (origin, destination, time) in one single utterance or in a different order.

In general, the dialog grammar approach has the following potential disadvantages

- The interaction may be experienced by a user as brittle, inflexible, and unforgiving, since it is difficult to support mix-initiative systems.
- Dialog grammars have difficulty with nonliteral language (indirection, irony, etc.).
- A speech act might be expressed by several utterances, complicating the grammar.
- A single utterance might express several speech acts, complicating the grammar.

To address these issues, more sophisticated approaches to enhance hand-built finite state dialog grammars have been attempted. For example, once can add statistical knowledge based on realistic data to dialog grammars. The statistical learning methods, like come CART, *n*-grams, or neural networks [3] can be used to learn the association between utterances and states in the training data.

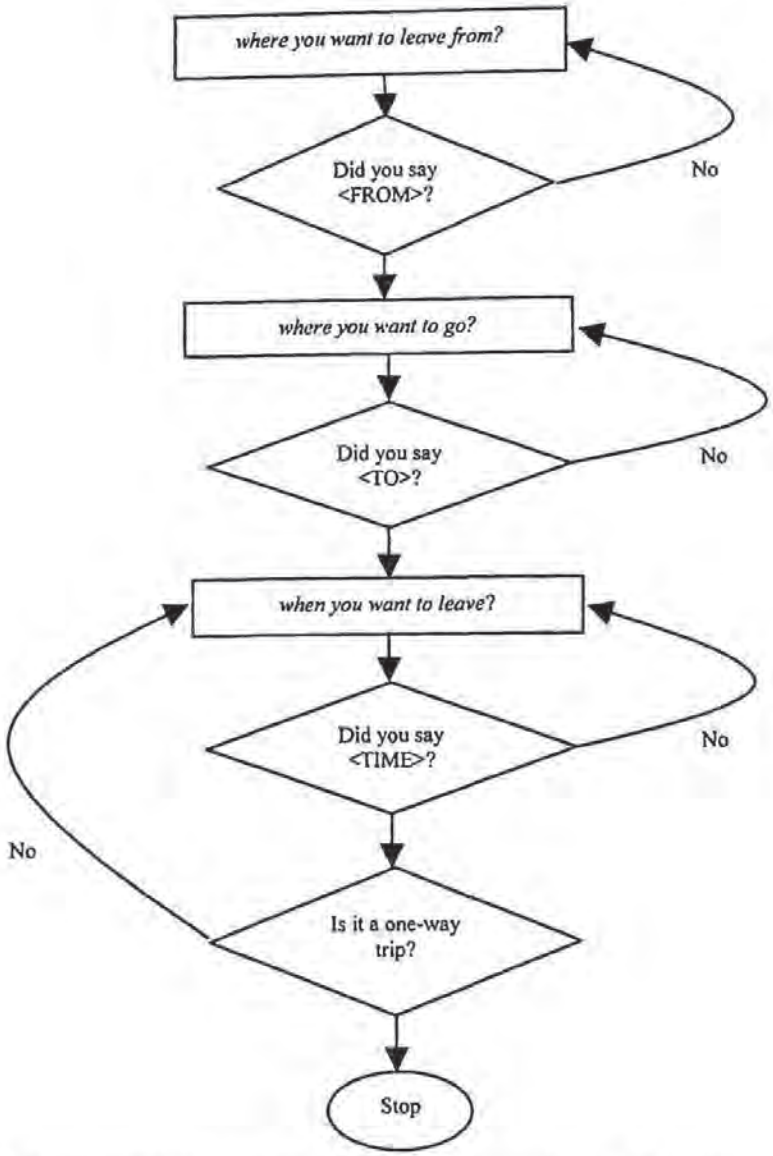



**Figure 17.19** A finite state dialog grammar for airline reservation (after [19]).

## 17.6.2. Plan-Based Systems

*Plan-based* approaches [2, 41] seek to overcome the rigidity and shallowness of dialog grammars and templates. They are based on the observation that humans *plan* their actions to achieve various *goals*. Thus, plans and goals are in some degree of correspondence. A system operating under these assumptions needs to *infer goals, construct and activate plans*. A user may have a preconceived plan for achieving his/her goals or may need to rely on the system to supplement or construct appropriate plans.

Plan-based systems are well studied in artificial intelligence (AI) [32, 65]. The mathematical foundation of the plan-based approach is inference. The behaviors of the system and the knowledge of the domain are programmed as a set of logical rules and axioms. The system interacts with the user to gather facts, which consequently trigger rules and generate more facts as the interaction progresses. As illustrated in Eq. (17.1), the goal of the dialog manager is to derive the action **A** based on discourse semantic $S_n$. Taking this view, the dialog manager is a natural outgrow of the semantic evaluation process. It is the step where the system's intent is computed. The outcome of the dialog manager is a message (via different rendering) the system conveys to the user.

In essence, a plan-based system is an embodiment of a state machine for which different discourse semantics are regarded as states. The difference, however, is that the *states* for the plan-based system are generated dynamically and not limited to a predetermined finite set. This capability of handling an unbounded number of states is a key strength of plan-based systems in terms of scalability.

Even a simple interaction can involve a variety of complex subgoals and pragmatic inferences. A partial plan for the airline reservation example in Section 17.6.1 is illustrated in Figure 17.20. One wants to know if a flight itinerary (F12) is an available one. The relationships among the goals and actions that compose a plan can be represented as a directed graph, with *goals*, *preconditions*, *actions*, and *effects* as nodes and relationships among these as arcs. These graphs illustrate the compositional nature of plans, which always include nested subplans, down to an almost infinite level of detail. The appropriate level of planning specification is thus a judgment call and must be application dependent.

The arcs are labeled with the relationship that holds between any two nodes. *SUB* shows that the child arc is the beginning of a subplan for the parent. At some point appropriate to the domain of the planning application, the SUBs will be suspended and represented as a single subsuming node. In Figure 17.20, ENABLE indicates a precondition on a goal or action. EFFECT indicates the result of an action. ENABLE indicates an enabling relationship between parent and child nodes.

Plan-based approaches incorporate a rich and deep model of rational behavior and, thus, in theory, permit a more flexible mode of interaction than do dialog grammar approaches. However, they can be complex to construct and operate in practice, due to reliance on logical and pragmatic inference, and due to the fact that no fully understood theoretical underpinning exists for their specification. The complexity of the domain of modeling often requires significant efforts from human experts to author the logical rules and axioms.

In plan-based theories of agent interaction, each dialog participant needs to construct and maintain a model of all participants' goals, commitments, and beliefs. Plans are, thus, a relatively abstract notion, leading to the hope that plans could be designed in an application-independent fashion, which would permit the development of *plan libraries*. Such libraries could be easily adapted to a variety of domains; just as specific entity models are derived from generic classes via inheritance in object-oriented programming.

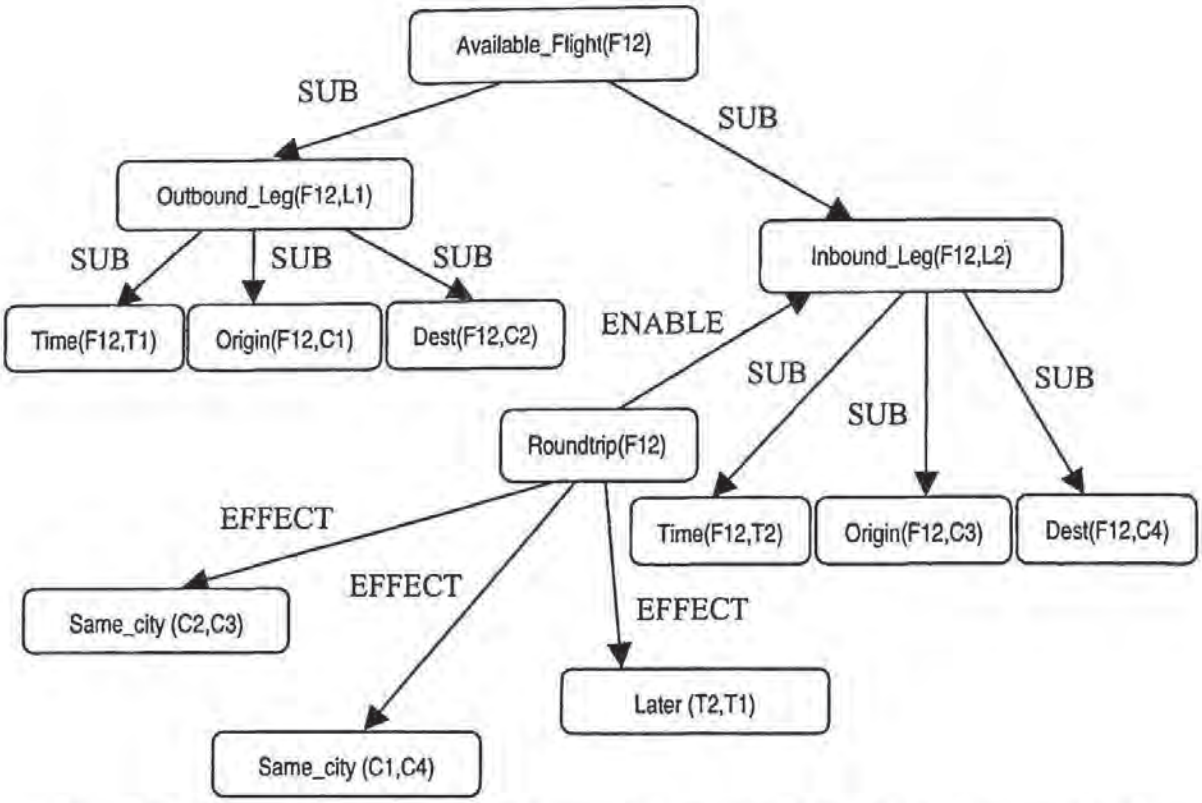



**Figure 17.20** A partial plan for the airline reservation example in Figure 17.19 represented as a graph.

The following operational cycle exemplifies the plan approach, describing interaction of two agents, X (the helpful assisting agent) and Y (the client). Interaction is stated from X's point of view [10].

- Observe Y's act(s)
- Infer Y's plan (using X's model of Y's beliefs and goals)
- Debug Y's plan, finding obstacles to success of plan, based on X's beliefs
- Adopt the negation of the obstacles as X's goal
- Plan to achieve those goals and execute the plan

A flight itinerary that at least contains an *Outbound_Leg* subgoal and another possible *Inbound_Leg* subgoal is a round trip. Let's assume F12 is a round trip itinerary. At the *Inbound_Leg* node, the interesting question is how much of the underlying goal (*Time*(F12,T2), *Origin*(F12,C3) and *Dest*(F12,C4)) can be inferred by the information provided by the system from the dialog so far, or from other known conditions. For example, the destination of the *Inbound_Leg* can be inferred from the origin in the outbound leg. The origin city can be inferred similarly. Going one step further, you can also infer that the de-

parture time for the inbound leg must occur after the departure time of the outbound leg (T1 < T2). Those three inferences are shown in the *Effect* arcs in Figure 17.20.

The goal inference could be a cooperative process, with the system making the minimal queries needed to verify and choose among alternative hypotheses. Or, it could be based on pure inference, with perhaps a confirmation step. Inference modeling can get very complicated. The technologies of inference are complex models of the beliefs, desires, and intentions of agents, making use of generic logical systems, which operate over the propositions corresponding to the nodes in a plan structure such as shown in Figure 17.20. Both user and system are assumed to be operating from partially shared world and discourse models consisting of beliefs about all relevant entities and their relationships. If utterances and speech acts are not in conflict with the constraints implied by the world models, communication and action can proceed. Otherwise, either the utterance itself must be further interpreted, supplemented, or clarified, or the world models need to be changed.

The natural expression of rational behavior, communication, and cooperation is some form of first-order logic. We define axioms and inference rules for *Belief* and *Intention*. If the modal operator for belief is $B$, axioms and inference rules for an agent $i$ with respect to proposition schemata $\phi$ or $\psi$ could be formalized in the following logical expression.

$$\begin{aligned}
&(B_i(\phi) \wedge B_i(\phi \Rightarrow \psi)) \Rightarrow B_i(\psi) \\
&B_i(\phi) \Rightarrow \neg B_i \neg \phi \\
&B_i(\phi) \Rightarrow B_i(B_i(\phi)) \\
&\neg B_i(\phi) \Rightarrow B_i(\neg B_i(\phi)) \\
&\neg B_i(\phi) \Rightarrow \neg B_i(B_i(\phi)) \\
&\forall x B_i(\phi) \Rightarrow B_i(\forall x \phi)
\end{aligned} \tag{17.5}$$

These describe appropriate conditions on beliefs of rational agents, such as entailment and consistency. Intentions, in turn, are formalized with respect to beliefs. For example, if an agent is to form an intention to bring about a state of affairs, it is reasonable that s/he believes this state of affairs is not currently in force:

$$I_i(\phi) \Rightarrow B_i(\neg \phi) \tag{17.6}$$

Other such axioms formalize related constraints on intentions, e.g., having an intention entails a commitment to achieving any preconditions, and belief in the possibility of doing so. Many more axioms involving all aspects of rational behavior, and formalizing, to some extent, the Gricean Maxims can be devised. For example, a kind of conversational cooperation occurs when a participant $i$ is willing to come to believe what $i$ believes his/her conversational partner $j$ is attempting to communicate (at least for the limited operational domains in question!), unless $i$ holds beliefs to the contrary:

$$B_i(I_j(B_i(\phi(j)))) \wedge \neg B_i(\neg \phi(j)) \Rightarrow B_i(\phi(j)) \tag{17.7}$$

When beliefs and intentions are modeled in this fashion, it may be possible to directly construct the core of a dialog engine based on rational principles as a theorem prover. Such a treatment is, however, beyond the scope of this discussion.

A few desirable system behaviors that would naturally follow from limited inference and goal tracking can be briefly examined. Unlike the dialog grammar approach, a plan-based system allows digression, since the user's intention model has been built into the plan. When a system is confused about a user's input, a cooperative system could begin to perform the critical pragmatic steps that uniquely distinguish the conversational interface. A chain of inferring the user's goal, based on the system's axioms, dialog history, and current knowledge, would be triggered.

It is essential for a system to track the *dialog focus*, or temporary centers of attention, in order to understand things that are unspoken but assumed to be salient across utterances. In this case, the user's input is ambiguous—*June 22* is for outbound or inbound flight? If the dialog architecture provides a method of tracking focus, it may be simple to resolve the legs from an earlier query.

Focus is a useful concept in dialog understanding. The basic idea is similar to the entity memory tracking in anaphora resolution (see Section 17.5.1.1)—at any given point in a conversational exchange, a few items are at the center of attention and are given preference in disambiguation. Other items are in the background but may be revitalized as centers of attention at some later point. A static area can be used to contain items that are assumed background knowledge throughout the exchange. The main goal of conversation can initialize the stack. As subgoals are elaborated, new focus sets are pushed on the stack, and when these subgoals are exhausted, the corresponding focus object is popped from the stack and earlier, presumably broader topics are resumed. Focus shifts that are not naturally characterized as refinements of a broader current topic may be modeled by initiating a new independent focus stack. Focus shifts may be cued by characteristic linguistic signals, such as cue words and phrases (*well now, ok!, by the way, wait!, hey*, etc.). In many cases, focus structure tracks the recursively embedded plan structures, such as that shown in Figure 17.20.

### 17.6.3. Dialog Behavior

Even though the behavior of the dialog manager is highly dependent on the domain knowledge and the applications, some general styles of dialog behavior are worth investigating. The first important dialog behavior is the *dialog initiative* strategies. System initiative systems have the advantage of narrowing the possible inputs from users, while paying the price for extreme inflexibility. Although user initiative strategy is often adopted for GUI-based systems, it is seldom implemented for SLU systems, since total flexibility is translated into high perplexity (resulting low system performance). For many applications, a flexible mixed initiative style is preferred over a rigidly controlled one. Although it is possible to implement a mixed initiative system using either dialog grammars or plan-based approach, the latter is more flexible because it can handling an unbounded number of states.

Most often, the response generated by the dialog manager is either a *confirmation* or a *negotiation*. Confirmation is important due to possible SLU errors. There are two major confirmation strategies—explicit or implicit confirmations. An *explicit* confirmation is a response solely for confirmation of what the system has heard. On the other hand, an *implicit*

confirmation is a response containing new input query and embedded confirmation with the hope that the user can catch and correct the errors if the embedded confirmation is wrong. The examples in Figure 17.21 illustrate both confirmation strategies.

SLU systems usually use a confidence measure as to when to use explicit and implicit confirmation. Obviously, explicit confirmation is used for low-confidence semantic objects while implicit confirmation is for high-confidence ones.

A negotiation response can arise whether a semantic object is fully filled or not. In the case of underspecification, there are some attributes of the semantic objects that cannot be inferred by the discourse manager. Possible actions range from simply pursuing the unfilled attributes in a predefined order, to gathering the entities in the knowledge base sorted by various keys. For cases of ill specification, an entity that matches the semantic object attributes does not exist. The planner can simply report such fact, or suggest removal or replacement of certain attributes, depending on how much domain knowledge is to be included in the planning process.

Often in the design process, we find it desirable to segregate a dialog into several self-contained sessions, each of which can employ specialized language, semantic, and even behavior models to further improve the system performance. Basically, these sessions are subgoals of the dialog, which usually manifest themselves as *trunk* nodes on the discourse tree. We implement a tree stack in which each trunk node is treated as the root for a discourse tree. The stack is managed in a first-in last-out fashion, as currently no digression is allowed from one subdialog to another. So far, the no-digression rule is considered to be a reasonable trade-off for dynamic model swapping.

Consider the example domain of travel itinerary planning [13]. At the top level is the *scenario*, which is the intended output of the interaction. The scenario is the entire itinerary, consisting of reservations for flights, hotels, rental cars, etc., all booked for the user at workable, coordinated times and acceptable prices and quality levels. A scenario might be: a flight out of the user's home city of Boston, from Logan airport, on April 2, at 4:00 PM on a particular flight, connecting in Dallas-Ft. Worth to another flight to a regional airport, an overnight hotel stay, a meeting the next day in the morning, a drive to a second local afternoon meeting, a flight from the regional airport in the evening to LA for a late meeting, another overnight stay in LA, a morning meeting at the hotel, and a return flight back to Boston later that same morning.

```
I:  I would like to fly to Boston.
R1: Do you want to fly to Boston?          (explicit confirmation)
R2: When do you want to fly to Boston?     (implicit confirmation)
```

**Figure 17.21** With the input *I would like to fly to Boston*, explicit confirmation response R1 *Do you want to fly to Boston*? only allows the user to confirm the destination, while implicit confirmation response R2 *When do you want to fly to Boston*? allows the user to provide departure-time information and have a chance to confirm the destination as well.

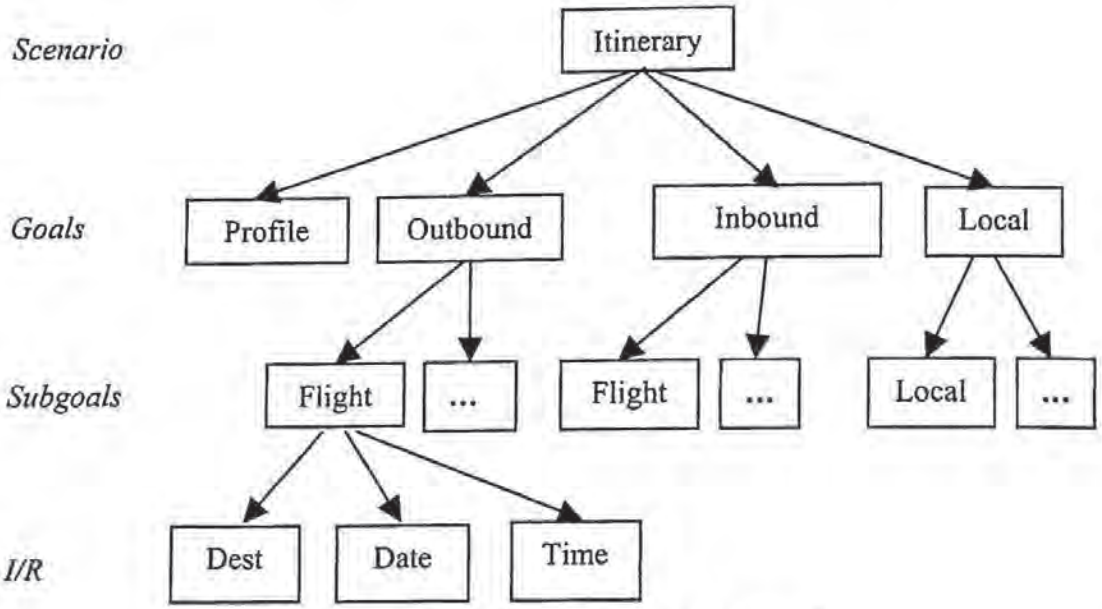



**Figure 17.22** A dialog structure hierarchy for travel.

Creating the finished itinerary for this scenario involves *goals*, generated by the system or the user. Goals might include: access user travel profile, book outbound and inbound flights, and make local arrangement (hotel reservation and car rental). Goals in turn may subsume *subgoals*. Subgoals are concerned with the details of planning. These would include establishing particular desired cities and airports for the flights, price investigation, queries about hotel location and quality, etc. The subgoals in their turn are generally realized via speech acts forming I/R pairs. A simplified schematic of the structure of the itinerary structure described above might appear as shown in Figure 17.22.

This structure lends itself to a variety of control mechanisms, including system-led and mixed initiative. For example, the system may ask guiding questions such as "*Where would you like to go*?" followed by "*What day would you like to leave?*" or the system could begin processing from the user's point of view by accepting an utterance like *I want to go from Boston to LA,* corresponding to the *Dest* node of a flight on the outbound flight, and responding with a query about the next needed item, e.g., *What day would you like to leave*? This system can also accommodate a user who may wish to talk about his or her hotel reservation immediately after making the outbound flight reservation, before arranging the inbound flight.

## 17.7. Response Generation and Rendition

*Response generation*, also known as the *message generation*, is the process in which the message is physically presented to the user. This is the stage that significantly involves human-factor issues, as discussed in Chapter 18. It is more susceptible to application-specific or user interface considerations. For example, to handle a message requesting the user to

select a sizable list of alternatives, a system with a suitable visual display might choose to present the whole list, while a speech-only system might require a more clever way. In this section we mainly focus on speech output modality and provide some thoughts on other popular output modalities.

A conversational interactive system requires a speech-output capability. The speech output may be comprised of system requests for clarification, disambiguation or repeat of garbled input; confirmation; prompting for missing information; statements of system capabilities or expectations; and presentation of results. At the lowest level, this is done via a text-to-speech engine, as discussed in Part III (Chapters 14, 15, and 16) and shown as a component in Figure 1.4. However, most text-to-speech engines have been designed for a read speech style. Moreover, such systems typically perform only shallow syntactic and semantic analysis of their input texts to recover some text features that may have prosodic correlates. Because the topic space of a task-oriented dialog system is narrower, there are opportunities to tune prosodic and other attributes of the speech output for better quality.

There are two major concerns in voice-response rendering. First is the creation or selection of the content to be spoken, and second is the rendition of it, which may include special prosodic markups as guidance to a TTS engine.

### 17.7.1. Response Content Generation

The response content can be explicitly tied to the semantic representation of the domain task and objects. The semantic class could incorporate custom prompts for specific slots or even for whole semantic classes. Whenever the dialog manager finds that specification for a particular slot is missing from a semantic object, it can check if it contains prompts. If prompts are present, one could be selected at random for presentation to the user.

Response prompts can be embedded in semantic representation. Prompts are usually provided for each slot to provide direction for users to fill the slot in the next dialog term. For example, the semantic class `ByName` defined in Figure 17.9 can be enhanced with the prompts in Figure 17.23.

Prompts could be associated with conditions. For example, in a flight information system, a conditional prompt can be inserted into the semantic class definition to inform users of the flight arrival time based on whether the flight has landed or not, as shown in Figure 17.24.

Other systems may include some categorization of prompts for different functions. For example, at the task level of an airline reservation system, the categorized message list might appear as shown in Figure 17.25. The grammar format makes provision for convenient authoring of messages that can be specified and accessed by functional type at runtime. The `BEMsg` is a special type of message. In this particular architecture, communication with the database engine (cf. the boxes application and database in Figure 17.2) is controlled by messages that are authored in the task specification. The URL attribute indicates a database

```
<!-- semantic class definition for ByName that has type
PERSON too -->
<class type="PERSON" name="ByName">
    <slot type="FIRSTNAME" name="firstname"
          prompt="Please specify the last name for [firstname]/>
    <slot type="LASTNAME" name="lastname"
          prompt="Please specify the first name for [lastname]/>   />
    <cfg>
          ..............
    </cfg>
</class>
```

**Figure 17.23** Semantic class ByName in Figure 17.9 is enhanced with prompts specified for the case of missing a particular slot information.

access. The </rclist> is the set of possible return codes from the back-end application (as it attempts to perform the specified command from the message). Again, every return condition is associated with a message by the task specification author. Those shown here include a simple confirmation of a successful completion, as well as a warning for *flight sold out* and a generic failure of transaction message.

```
<class type="FLIGHT" name="Flight">
    <slot type="FLIGHTNO" name="flight_no">
    <slot type="TIME" name="sch_time">
    <slot type="TIME" name="actu_time">
    <slot type="CITY" name="dep_city">
    <slot type="CITY" name="arr_city">
    <slot type="AIRLINE" name="airline">
    <prompt condition= "$SYS_TIME > [actu_time]">
          Flight [flight_no] is landed at [actu_time]
    </prompt>
    <prompt condition= "default">
          Flight [flight_no] is scheduled to land at [sch_time]
    </prompt>
    <cfg>
          ..........
    </cfg>
</class>
```

**Figure 17.24** A semantic class Flight contains a conditional prompt to inform users when invalid [depart_time] is detected.

```
<messages>
  <msg id="Help"> Please specify the flight time, origin and destination </msg>
  <msg id="Cancel"> Canceling itinerary... </msg>
  <msg id="Confirm"> Buying ticket from [origin] to [dest] on [time]? </msg>
  <msg id="BEMsg" url="http://server/...?op=buy&time=[time]&flight=[flight]..">
    <rclist>
       <rc id="OK"> Complete buying </rc>
       <rc id="SO"> The flight is sold out </rc>
       <rc id="ERROR"> Cannot complete transaction </rc>
    </rclist>
  </msg>
</message>
```

**Figure 17.25** An example of categorization of prompts for an airline reservation SLU system.

Such systems can incorporate other kinds of categorization as well. For example, a system might provide a battery of responses to a given task or subtask situation, varying depending on a speech recognition confidence metric. Thus a set of utterances ordered by decreasing confidence might appear as:

```
You want to fly to Boston?
Did you say Boston?
Could you repeat that, please?
Please state a flight reservation.
```

Systems of this type are sometimes referred to as *template systems* for response generation. They have the advantages of direct authoring and simplicity of implementation and may provide very high quality if the message templates of the application can be played with matching digitized speech utterances or carrier phrases in the synthesizer.

The specificity and application-dependent qualities of template-based systems are sometimes perceived as weaknesses that could potentially be overcome by more general, flexible, and intelligent systems. In these systems the *message generation* box could subsume discrete modules, as shown in Figure 17.26. The semantic representation would typically be akin to logical forms (see Chapter 2) expressed via semantic frames or conceptual graphs. The representation would include abstract expression of content as well as speech-act type and other information to guide the tactical or low-level aspects of utterance generation, such as word choice, sentence type choice, grammatical arrangement, etc.

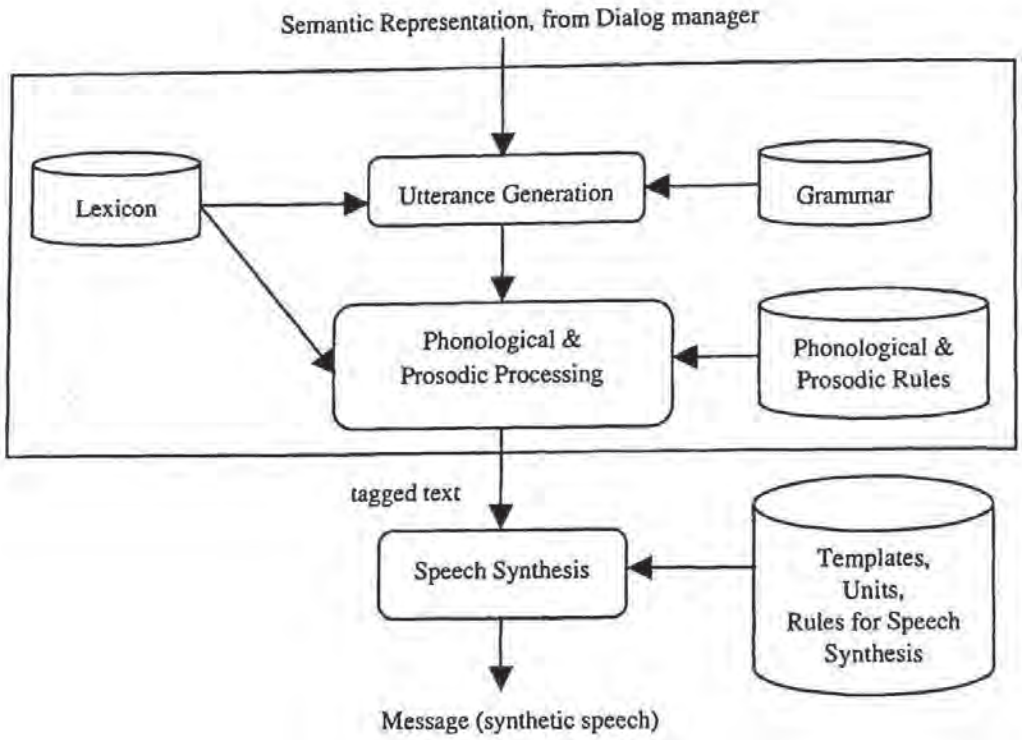



**Figure 17.26** Natural language generation and rendition modules.

Natural language generation from abstract semantic input is a deep and complex field. Let us briefly consider a slightly more abstract form of template-selection mechanism that could gracefully either accommodate a simple set of static, authored response utterances or, alternatively, serve as a form of semantic input to a generalized, NLP-based utterance generation module. Imagine that instead of simply providing lists of prompt strings with embedded slot identifiers, a system of parameterization can be used [24]. The parameters could be at varying levels of abstraction and would function as descriptors of static content when preauthored prompts were being used, or would serve as a kind of input semantic representation when a general natural language was used. The set of parameters might include attributes of utterances such as the following:

- *Utterance type*: mood of the sentence, i.e., declarative, *wh*-question, yes/no question, or imperative.
- *Dialog or speech act*: confirmation, suggestion, request, command, warning, etc.
- *Body*: some characteristic lexical content for the utterance, apart from any situation-dependent words and concepts. This could serve as a hint to a generator. In many cases this would be the main verb of a sentence and might

also include characteristic cue words, especially for functional transitions, e.g., *however*, *now*, etc.

- *Given*: information that is understood from the discourse history. This is usually represented as pronouns or other anaphora in the generated utterance.
- *New*: anything that is in the informational foreground, due to lack of prior mention, but may not be precisely the purpose of the prompt, per se. New material typically receives some kind of prosodic prominence in speech.

Examples of these parameter indices for templates from a theater ticket-reservation domain might appear as in Table 17.4. The basic idea of the parametric approach is that such a level of medium abstraction allows for flexibility in the choice of deployment tactics. If a full set of static prompts and response utterances is available for all cases, then this approach reduces to a template system, though it does provide the potential for separation of grammars and prompt files. If, however, a natural language generation component is available for dynamic message generation, a parameter set like that above can serve as input.

**Table 17.4** Sentence generation indices for an airline reservation SLU system.

| Act | Type | Body | Given | New | Example |
|---|---|---|---|---|---|
| Meta [sorry] | Decl | *no* | - | - | *No, sorry.* |
| Verify | Y/N-Q | | | Boston | *Boston?* |
| Request-info | WH-Q | *fly* | you | thing | *When do you want to fly?* |
| Request-info | WH-Q | *want* | you | airline tomorrow | *Which airline would you like to fly tomorrow?* |
| Stmt[sorry] | Decl | *sold out* | it | - | *Sorry it is sold out.* |
| Stmt | Decl | *sold out* | USAir | - | *Sorry, USAir is sold out.* |

## 17.7.2. Concept-to-Speech Rendition

Once the response content is generated, the SLU system needs to render it into a waveform to play to the users. The task is naturally assigned to a text-to-speech component. However, the response generated in the previous session is more than text message. It contains the underlying semantic information, because it is usually embedded in the semantic representation as shown in Figure 17.23 and Figure 17.24. This is why the speech rendition is often done through a *concept-to-speech* module. A concept-to-speech system can be considered as a text-to-speech system with input text enhanced with domain knowledge tags. With these extra tags, a concept-to-speech system should be able to generate tailored speech output to better convey the system intention.

Chapter 15 discussed the role of prosody in human perception. When messages are generated, it is expected that they are supplemented with hints as to their information structure. At a minimum, the message generation component can identify which parts of the utterance constitute the *theme*, which is material understood, previously mentioned, or

somehow extending a longer thread of coherence in the dialog, from the *rheme*, which is the unique contribution of the present utterance to the discourse [36]. If such a distinction is marked on the generated utterances, or templates, it can be associated with characteristic pitch contour, prosodic phrasing, and other effects (see Chapter 15).

For example, in the question-answer pair shown in Figure 17.27 (from ordinary human conversation), the theme and rheme components are bracketed. The theme of the answer consists of a mention of Mary, and the act of driving, both carried forward from the question. The theme consists of new information, the answer to the question, embedded in a kind of placeholder noun phrase. Clearly, the input to the message generation component requires some indication of which entities of the input semantic representation are linked to discourse history.

```
Q: Which car did Mary drive?
A: (Mary drove)th (the RED car.)rh
```

**Figure 17.27** A question-answer pair with theme and rheme components marked.

Prosodic rules are triggered by information structure. In general, a theme in the early part of a statement may be realized with a rise-fall-rise pitch contour, often with turning points in the contour aligned with lexically stressed or other salient syllables of the words in the theme. Rheme marking by pitch contour is also essential for naturalness, and a common rheme tune in English declaratives is a slight rise up to the final lexically stressed syllable, followed by a fall to the bottom of the speaker's pitch range. The actual alignment of pitch extrema will depend on the position of focus, or maximum contrast and information value, within either the theme or the rheme.

In Figure 17.27, the word *RED* is in focus within the rheme. If the question had implied a contrast between Mary's car and other people's cars, it would be acceptable to esta b-lish a focus on *Mary* in the theme as well, marked by a pitch accent (see Chapter 15). Sometimes the portion of either theme or rheme that is not in focus (e.g., *drove* or *car*) is called the *ground* [54, 55].

The response generator could add such rheme-theme information that may be used to trigger more specialized prosodic rules. For example, one experimental system is based on a message generator that dynamically creates concise descriptions of individual museum objects during a tour, while attempting to maximize correlations to objects a museum visitor has already seen [21]. During the response generation phase, simple entities and factual statements are combined, first into a semantic graph and then into a text, in which the rhetorical functions of utterances and clauses, and their relations to one another, are known. This information can be passed along to a synthesizer in the form of markup tags within the text. A synthesizer can then select appropriately interesting pitch contours that indirectly reflect rhetorical functions.

In a dialog system, other attributes beyond rheme-theme kinds of information structure, such as speech-act type, may have characteristic intonation patterns. This might include a regretful-sounding contour (perhaps sampled from real speaker data) applied when apologizing (*Sorry, that flight is sold out*) or a cheerful-sounding greeting. Although the concept-to-speech module can be implemented as just a text-to-speech system that take the advan-

tage of the extra semantic knowledge to generate appropriate prosody, the most natural speech rendition is still to play back a prestored waveform for the entire message. This is why the concept-to-speech module usually relies heavily on playback of template waveform. However, it is obvious that we can't record every possible message like "*Flight [flight_no] is schedule to land at [sch_time]*" in Figure 17.24. Instead, a carrier sentence can be recorded and the slots can then be replaced with real information. The slot can be synthesized with an adapted TTS, which essentially eliminates the need for a front end in the TTS system.

One problem of this approach is that the same prosody is used for a word regardless of where it appears, which results in lower naturalness, because prosodic context is important for natural speech. Enhanced quality can be achieved by having different instances of those slot words, depending their contexts. For example, we can have different *one* recordings depending on whether it is the first digit on a flight number, the second, or the last. Determining the number of different contexts where a slot needs to be recorded is typically done much like the context-dependent acoustic modeling discussed in Chapter 9. This technique increases the naturalness, at the expense of increasing the number of necessary recordings.

### 17.7.3. Other Renditions

So far, we have assumed that a dialog system may be used only in a speech-only modality. Although such systems have found many applications, multi+modal interaction may be more compelling, as discussed in Chapter 18. In fact, voice output might not be the best information carrier in such an environment. For example, the latest wireless phones are equipped with an LCD screen that allows for e-mail and Web access. If a high-resolution screen is available, the renditions mechanism will likely be visually oriented.

When renditions become visually oriented, the message generation component needs to be replaced by a graphic display component. Since GUI has been the dominant platform for deploying major computer applications today, the behavior and technique of such a display component is well studied and documented [17]. The SLU system needs only to pass the semantic representation from the dialog management module to a GUI rendering module. Of course, the GUI rendering module should also be equipped with domain knowledge to generate best rendering to convey the dialog message. MiPad [22] is such an example and is discussed in Chapter 18.

## 17.8. EVALUATION

How do we define a quantitative measure for understanding? Evaluation of understanding and dialog is a research topic on its own. We review a number of research techniques being pursued.

### 17.8.1. Evaluation in the ATIS Task

An application used for development, testing, and demonstration of a wide variety of dialog systems is the Air Travel Information Service (ATIS) task, sponsored by the DARPA Spoken Language Systems program [20]. In this task, users ask about flight information and

make travel arrangements. To enable consistent evaluation of progress across systems, a corpus of data for this task has been collected and shared among research sites.

The application database contains information about flights, fares, airlines, cities, airports, and ground services, organized in a relational schema. Most user queries, though they may require some system interaction in order to specify fully, can be answered with a single relational query. The ATIS data collection is done using the wizard-of-oz framework.[4] A user interacts with the system as though working with a fully automated travel planner. Hidden human *wizards* were used in the data-collection process to provide efficient and correct responses to the subjects. A typical scenario presented as a task for a subject to accomplish by means of the automated assistant is as follows:

> Plan the travel arrangements for a small family reunion:
>
> First pick a city where the get-together will be held. From three different cities (of your choice), find travel arrangements that are suitable for the family members who typify the *economy*, *high class*, and *adventurous* life styles.

After data collection, each query was classified as context dependent or context independent. A context-dependent query relies partially on past queries for specification, such as "*Is that a non-stop flight?*" Many of the system tests based on ATIS require not only accuracy of speech recognition (the user's spoken query), but also semantic interpretation sufficient to construct an SQL query to the database and correctly complete the desired transaction. Evaluation of ATIS was based on three benchmarks: SPREC (speech recognition performance), NL (natural language understanding for text transcription of spoken utterances), and SLU (spoken language understanding). For SLU systems we are interested only in the last two benchmarks.

With the help of constrained domain of ATIS, correct understanding can be translated into correct database access. Since database access is usually done via SQL database query, the evaluation of understanding can be performed in the domain of generated SQL queries. However, it is still ambiguous when someone would like to query flights around 11:00 a.m. For the purpose of understanding, how wide a time frame is *around* considered to be?

Many examples of queries contain some ambiguities. For instance, when querying about the flights between city X and Y, should the system display only the flights from X to Y; or flights in both directions. To alleviate the ambiguity, each release of ATIS training corpus was accompanied by a *Principles of Interpretation* document that has standard definitions of the meaning of such terms like *around* (means within a 15-minute window) and *between* (means only *from*).

Once the correct understanding is represented as an SQL query, ATIS can be easily evaluated by comparing the SQL queries generated by SLU systems against the standard labeled SQL queries. The utterances in ATIS are classified into three types:

[4] The wizard-of-oz data collection framework is described in Chapter 18.

- A—semantically independent of earlier utterances, so per-turn semantic interpretation can uniquely identify the semantic intent.
- D—semantically dependent upon earlier utterances, so discourse knowledge is required to provide full interpretation.
- X—unevaluatable, so a response such as *No answer* or *I don't understand you, could you repeat yourself* is considered a right answer.

The other debatable item is whether a *No answer* output for type A and D utterances should be treated equally as a false SQL query. In the original 1991 ATIS evaluation, a false SQL query for type A and D utterances is penalized twice as heavily as a *No answer* output for type A and D utterances. However, the decision was dropped for the 1993 ATIS evaluation. ATIS decided not to evaluate dialog component for three reasons. First, dialog alters users' behavior during data collection. Users' utterances are highly contingent on the performance of the wizard-of-oz system, so the data collected has little use for systematic training and testing. Second, the SLU systems would likely have to be tested by real subjects. Third, the evaluation of dialog behavior is highly subjective, since effectiveness and user friendliness are generally vaguely defined.

### 17.8.2. PARADISE Framework

The evaluation of a dialog system is subjective in nature and is typically done in an end-to-end fashion. In such a framework, objective criteria like number of dialog turns and system throughput, and subjective measures like user satisfaction, are typically used.

One of the most sophisticated systems for evaluating dialog systems ever developed is the PARAdigm for Dialog System Evaluation (PARADISE) [57]. The designers of this framework took a comprehensive view of the many potential factors affecting dialog evaluation, in particular the distinction between measuring success of transaction (quality) and cost of the dialog, both in human and system terms. A decision-theoretic method, as shown in Figure 17.28, is used to explicitly weight these various disparate factors to achieve a unified measure. In addition, the PARADISE metrics can derive discrete scores for subdialogs, which is useful for diagnosis, comparison across systems, and tuning.

A simple measure for task success can be the following question: "*Was all the needed information exchanged, in the correct direction (user to system, system to user) at each step*?" PARADISE provides a framework for defining, for any interaction in a limited domain, a simplified representation of the minimal required information and its directionality. In PARADISE terms, this is an *attribute-value matrix* (AVM) showing the names and instantiations of required elements at dialog completion. This could be derived from reference frames for each required concept in a dialog exchange, with mandatory slots marked for legal completions. Once such reference frames or matrices are available, different dialog strategies that address the same function can be compared over many instantiations (test dialog sessions), using statistical measures that assess confusability and length.

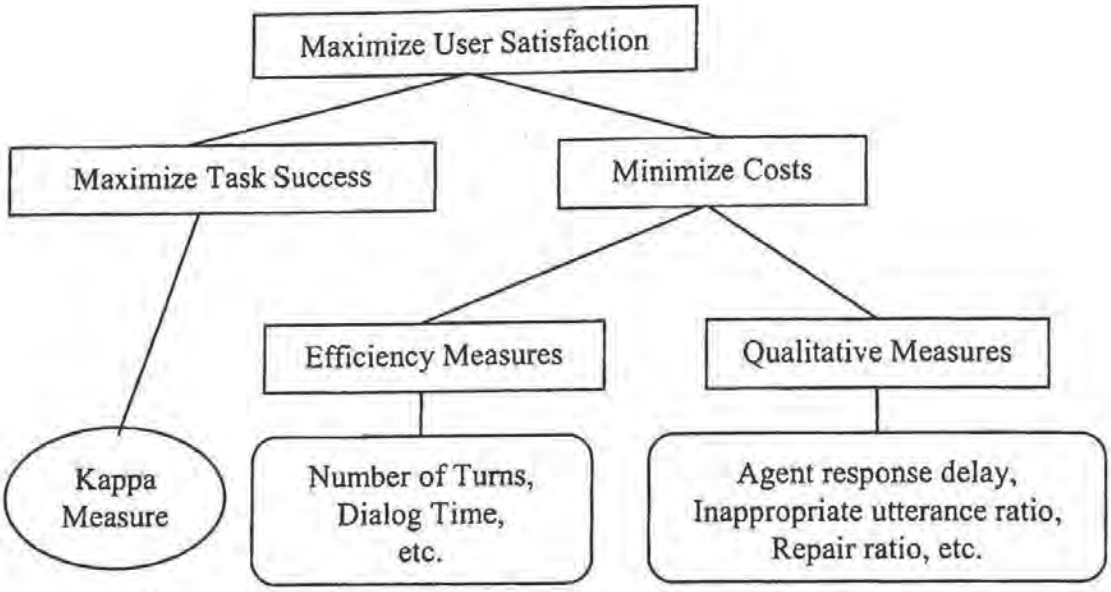



**Figure 17.28** PARADISE's structure of objectives for spoken dialog performance [57].

For example, imagine an ATIS-like application that had the following information attributes, with the possible values listed in Table 17.5. An utterance such as "*I want to go from Torin to Milan*" communicates legal DC and AC attribute values from user to system. This is a limited-domain system by assumption, so confusions are assumed to occur within the possible values of the application. For example, if the system instantiates the *Depart-City* (DC) slot with *Trento* instead of *Torin* after processing the given sample utterance, it is a confusion that can be recorded in a confusability matrix over all dialog test sessions. A subsection of such a possible confusability matrix, covering only the DC and AC attributes, is shown in Table 17.6, which shows only confusion within an attribute type that covers a consistent vocabulary (city names, instantiating the DC and AC attributes). In practice, however, the full matrix might show confusions across attribute types, such as *morning* for *Milan*, etc.

Given a confusability matrix $M$ over all possible attributes in the application, we can apply the Kappa coefficient [48] to measure the quality characterizing the task's success at meeting the information requirements of the application:

**Table 17.5** Attribute-value table [57].

| Attribute | Possible Values |
|---|---|
| Depart-City (DC) | Milan, Rome, Torin, Trento |
| Arrival-City (AC) | Milan, Rome, Torin, Trento |
| Depart-Range (DR) | Morning, evening |
| Depart-Time (DT) | 6am, 8am, 6pm, 8pm |

**Table 17.6** Confusability matrix for city identification [57].

| | Depart-City | | | | Arrival-City | | | |
|---|---|---|---|---|---|---|---|---|
| Data | Milan | Rome | Torin | Trento | Milan | Rome | Torin | Trento |
| Milan (depart) | **22** | | 1 | | 3 | | | |
| Rome (depart) | | **29** | | | | | | |
| Torin (depart) | 4 | | **16** | 4 | | | 1 | |
| Trento (depart) | 1 | 1 | 5 | **11** | | | 1 | |
| Milan (arrive) | 3 | | | | **20** | | | |
| Rome (arrive) | | | | | | **22** | | |
| Torin (arrive) | | | 2 | | 1 | 1 | **20** | 5 |
| Trento (arrive) | | | 1 | | 1 | 2 | 8 | **15** |
| sum | 30 | 30 | 25 | 15 | 25 | 25 | 30 | 20 |

$$\kappa = \frac{P(A) - P(E)}{1 - P(E)} \tag{17.8}$$

where $P(A)$ is the proportion of times that the AVMs for the actual set of dialog agree with the AVMs for the interpreted results, and $P(E)$ is the proportion of times that AVMs for the dialog and interpreted results are expected to agree by chance. $P(E)$ can be estimated by $P(E) = \sum_{i=1}^{n} \left(t_i/T\right)^2$, where $t_i$ is the sum of the frequencies in column $i$ of $M$ and $T$ is total frequencies $(t_1 + \cdots + t_n)$ in $M$. The measure of $P(A)$ (how well or poorly the application did in information extraction) is calculated simply by examining how much of the total count occurs on the diagonal: $P(A) = \sum_{i=1}^{n} M(i,i)/T$.

In addition to task success, system performance is also a function of several cost measures. Cost measures include efficiency measures, such as the number of dialog turns or task completion time; as well as qualitative measures, such as style of dialog or how good the repair mechanism is. If a set of test dialogs is available, with experimentally measured user satisfaction (the predicted categories), the kappa measure, and quantitative measures of cost (denoted as $c_i$, such as counts of repetitions, repairs etc.), linear regression can be used, over the z-score normalization of these predictor terms, to identify and weight the most important predictors of satisfaction for a given system. Thus, the performance can be defined as:

$$\text{Performance} = \alpha * \mathfrak{A}(\kappa) - \sum_{i=1}^{n} w_i * \mathfrak{A}(c_i) \tag{17.9}$$

where $\mathfrak{A}$ is the z-score normalization function $\mathfrak{A}(x) = \frac{x - \bar{x}}{\sigma_x}$.

Evaluating a dialog system involves having a group of users perform tasks with ideal outcomes. Then the cost measures and task success kappa measure are estimated. These measures are used to derive the regression weights in Eq. (17.9). Once the regression weights are attained, one could possibly predict the user satisfaction when a subpart of the dialog system is improved.

## 17.9. CASE STUDY—DR. WHO

Dr. Who is a project at Microsoft Research on its multimodal dialog system development. It incorporates many of the dialog technologies described in this chapter. We use Dr. Who's SLU engine as an example to illustrate how to effectively create practical systems [22, 58–61]. It follows the mathematical framework illustrated in Eq. (17.1). The system architecture is shown in Figure 17.29. Since it intends to serve as a general architecture for multimodal dialog systems, it makes some simple assumptions at the architecture level. First, it replaces the *speech recognizer* and *sentence interpretation* modules with a *semantic parser* for each modality. The *response rendering* is merged into *dialog manager* with different XSL style sheets for each media output.

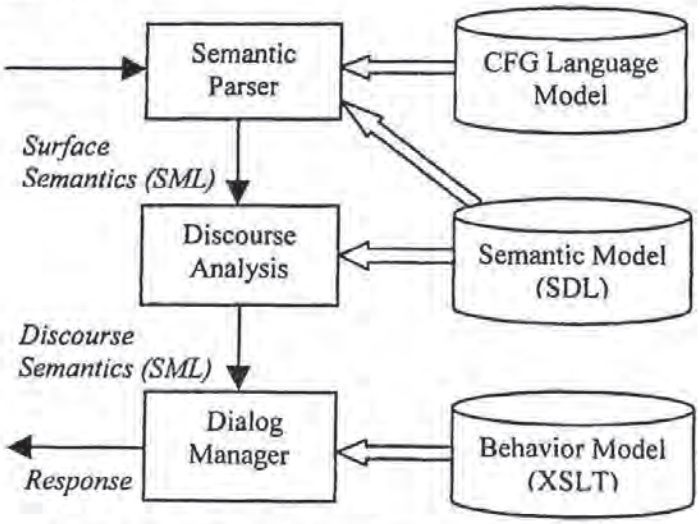



**Figure 17.29** The Dr. Who system architecture [60].

### 17.9.1. Semantic Representation

Semantic representation is a critical part in Dr. Who's SLU engine design. Essentially, the semantic objects are an abstraction of the speech acts, the domain knowledge, and the application logic. They are designed to encapsulate the respective language models and dialog actions that govern their creation and behaviors. The system components communicate with one another through events surrounding the semantic objects. In this view, the dialog (including logic inferences) is an integral part of the discourse semantic evaluation process.

There are two types of semantic objects in Dr. Who. The first type is the *functional* semantic object that is used to represent linguistic expressions in the user's utterance. The

second type is the *physical* semantic object that is used to represent real-world entities related to the application domain. Both types of semantic objects are represented by semantic frames and specified in the semantic markup language (SML), which is an extension of XML. Following the principles of the XML schema, Dr. Who defines the schema of SML in another XML called semantic definition language (SDL). SDL is designed to support many discourse and dialog features. In addition, SDL is suited to represent the domain knowledge via the application schema, the hierarchy of the semantic objects, and the semantic inference rules.

The format of various semantic classes follows SDL representations in Dr. Who. The terminal and nonterminal nodes on the parse are denoted in SDL with tags `<verbatim>` and `<class>`, respectively. These tags refer to the semantic objects and have the *name* and *type* attributes. The `type` attribute corresponds to the entity type the semantic object eventually would be converted to; it plays a key role in inheritance and polymorphism, as described in Section 17.3.1. When a semantic object is unique in its type, SDL can automatically assume its type as the name. In addition, SDL defines a `<cfg>` tag for the language model that governs the instantiation of a semantic object ,and the language model could be stored in another file. An `<expert>` tag can be defined for the system resource to physically convert a semantic object to a domain entity. Finally, the tag `<slot>` in SDL defines the descendant for a nonterminal node.

Take the semantic class for Microsoft employee directory as an example. The simple application answers queries on an employee's data such as office location, phone number, hiring date, etc. An item that can be asked is a semantic terminal `DirectoryItem` as defined in Figure 17.30. To allow users to ask more than one directory item at one dialog turn, a multiple semantic class `DirectoryItems` is also defined recursively, as shown in Figure 17.30.

```
<verbatim type="DirectoryItem" …>
   <prod name="office"/>
   <prod name="phone"/>
   <prod name="hiring date"/>
   …
</verbatim>
<class type="DirectoryItems" …>
   <slot type="DirectoryItem"/>
   <slot type="DirectoryItems"/>
   <cfg ref="DirectoryItems.cfg"/>
</class>
```

**Figure 17.30** The terminal semantic class `DirectoryItem` and nonterminal semantic class defined in Dr. Who using SDL. Note that the definition of `DirectoryItems` contains a recursive style, which can accommodate more than one `DirectoryItem` [59].

The <prod> tags inside a terminal semantic object indicate that the terminal is of an enumeration type, and all the possible values are text normalized to the string values of the *name* attribute. The main speech act, the query, is modeled by the functional semantic class DirectoryQuery, as shown in Figure 17.31.

```
<class type="DirectoryQuery" ...>
    <slot type="Person"/>
    <slot type="DirectoryItems"/>
    <expert clsid="..."/>
    <cfg ref="Directory.cfg"/>
</class>
<include ref="PeopleGrammar.sdl"/>
```

**Figure 17.31** The main semantic class DirectoryQuery defined in Dr. Who using SDL [59].

The semantic object can be instantiated following the language model in "Directory.cfg" and, once instantiated, is handled by a system object identified by its class id (clsid). The system object then formulates the query language that retrieves the data from the database. It is also possible to embed the XML version of the query language (e.g., XQL) within the <expert> tag. Semantic models can be nested and reused, as shown in the <include> tag in the above example, where the semantic model for people is referred.

## 17.9.2. Semantic Parser (Sentence Interpretation)

For speech modality, Dr. Who employs a speech recognizer with unified language models [62] that take advantage of both rule-based and data-driven approaches, as discussed in Chapter 11. Once we have text transcription of user's utterances, a robust chart parser [61] similar to the one described in Section 17.4.1 is used for sentence interpretation.

The emphasis of sentence interpretation is to annotate the user's utterance in a meaningful way to generate functional semantic entities. Essentially, the surface SML represents a semantic parse. Thus, after a successful parse, the corresponding surface semantic objects are instantiated based on the semantic classes whose CFG grammars are fired. While in SDL we use static tags such as <class> and <verbatim> for the semantic classes, the instances of a semantic object use the object name as the tag in SML. For example, the surface SML for an utterance "*What is the phone number for Kuansan*" is shown in Figure 17.32.

```
<DirectoryQuery ...>
    <PersonByName type="Person" parse="kuansan">
        Kuansan
    </PersonByName>
    <DirectoryItem type="DirectoryItem" parse="phone num-
ber">
        phone
    </DirectoryItem>
</DirectoryQuery>
```

**Figure 17.32** The surface semantic object DirectoryQuery represented in SML after a successful parse [59].

### 17.9.3. Discourse Analysis

As mentioned in Section 17.5, the goal of discourse analysis is to resolve surface semantic objects to discourse semantic objects. For the surface semantic object in Figure 17.32, the discourse engine binds the three semantic objects (i.e., the person, the directory item, and the directory query itself) to real-word and functional entities represented in the SML example, as shown in Figure 17.33.

```
<DirectoryQuery ...>
   <Person id="kuansanw" parse="kuansan">
      <First>Kuansan</First>
      <Last>Wang</Last>
      ...
   </Person>
   <DirectoryItem parse="phone number">
      <phone>+1(425)703-8377</phone>
   </DirectoryItem>
</DirectoryQuery>
```

**Figure 17.33** The discourse semantic objects for the surface semantic object illustrated in Figure 17.32 [59].

Note that the parse string from the user's original utterance is kept so that the rendering engine can choose to rephrase the response using the user's wording.

When an error occurs, the semantic engine inserts an `<error>` tag in the offending semantic objects with a code indicating the error condition. For example, if the query is for a person named *Derek*, the discourse SML might appear as shown in Figure 17.34.

```
<DirectoryQuery status="TBD" focus="Person" ...>
   <PersonByName type="Person" parse="Derek" status="TBD"...>
        <error scode="1" count="27"/>
        <Person id="derekba">
           <First>Derek</First>
           <Last>Baines</Last>
           ...
        </Person>
        <Person id="dbevan">
           <First>Derek</First>
           <Last>Bevan</Last>
           ...
        </Person>
           ...
   </PersonByName>
           ...
</DirectoryQuery>
```

**Figure 17.34** A discourse semantic object in Dr. Who contains an `<error>` tag indicating the error condition [59].

In Figure 17.34, semantic objects that cannot be converted (e.g., `DirectoryQuery` and `PersonByName`) are flagged with a status "TBD". Discourse SML also marks the dialog focus, as in the `DirectoryQuery`, that indicates the places where the semantic evaluation process fails to continue. These two cues assist the behavior model in deciding the appropriate error-repair responses.

Dr. Who uses three priority types of entity memory (discourse memory, explicit, and implicit turn memory) to resolve relative expressions. Anaphora and deixis are treated as common semantic classes, so they can be resolved according to the algorithm described in Section 17.5.1.1. Ellipsis is treated as an automatic inference. Unless marked as NO_INFER in the semantic class definition, every slot in a semantic class can be automatically inferred. The strategy to automatically resolve partially specified entities is as follows.

During the evaluation stage, a partially filled semantic object is first compared with the entities in the three-entity memory based on the type compatibility. If a candidate is found, the discourse analysis module then computes a goodness-of-fit score by consulting the knowledge base and considering the position of the entity in the memory list. The semantic object is converted immediately to the entity from the memory if the score exceeds the threshold. In the process, all the actions implied by the entities are carried out following the order in which the corresponding semantic objects are converted. For example, the second user's query in the dialog illustrated in Figure 17.35 contains an ellipsis reference to `DirectoryItem` *`office`*, which can be resolved using the discourse entity memory.

*U: Where is his office?*
*S: The office is in building 31, room 1362.*
*U: How about Kuansan's?*
*S: The office is in building 31, room 1363.*

**Figure 17.35** A dialog example in the Dr. Who system. The second user's query contains an ellipsis reference to `DirectoryItem` *`office`* [59].

## 17.9.4. Dialog Manager

To support mixed-initiative multimodal dialogs, Dr. Who employs a plan-based approach instead of dialog grammars. The dialog manager that handles dialog events surrounding semantic objects is very similar to a GUI program that handles GUI events surrounding graphical objects. These events can be handled synchronously or asynchronously based on various implementation considerations. In addition, the design enables a seamlessly integrated GUI and speech interface for multimodal applications to embrace the same human-computer interaction model.

Dr. Who SLU engine can use XSL-transformations (XSLT) [59] for specifying the behavior of a plan-based dialog system. XSLT, a recent World Wide Web Consortium (W3C) standard, is a specialized XML intended for describing the rules of how a structured document in XML can be transformed into another, say in a text-to-speech markup language for speech rendering or the hypertext markup language (HTML) for visual rendering. Its core construct is a collection of predicate-action pairs: each predicate specifies a textual pattern in the source document, and the corresponding action will produce a text segment in the

output whenever the pattern specified by the predicate is seen in the source document. The output segment is specified through a programmable, context-sensitive template. XSLT defines a rich set of logical controls for composing the templates. The basic programming paradigm bears close resemblance to a logical programming language, such as Prolog, which facilitates logic inference in plan-based systems. As a result, XSLT possesses sufficient expressive power for implementing crucial dialog components, ranging from defining dialog plans, realizing dialog strategies, and generating natural language, to manipulating prosodic markup for text-to-speech synthesis and creating dynamic HTML pages for multimodal applications.

Assuming TTS output, the planning rules that render the discourse SML of Figure 17.33 in text can be expressed in XSLT as shown in Figure 17.36.

```
<xsl:template match="DirectoryQuery[@not(status)]">
    For <xsl:apply-templates select="Person"/>, the
    <xsl:apply-templates select="DirectoryItem"/>.
</xsl:template>
<xsl:template match="Person">
    <xsl:value-of select="First"/>
    <xsl:value-of select="Last"/>
</xsl:template>
<xsl:template match="DirectoryItem">
    <xsl:apply-templates/>
</xsl:template>
<xsl:template match="phone">
    phone number is <xsl:value-of/>
</xsl:template>
```

**Figure 17.36** A TTS response-rendering rule for discourse SML of Figure 17.33. This rule generates a text message "*For Kuansan Wang, the phone number is +1(425)703-8377*" [59].

This rule leads to a response *For Kuansan Wang, the phone number is +1(425)703-8377*. Elaborated functions, such as prosodic manipulations in text to speech markup, can be included accordingly. To change the output to Web presentation, the above XSLT style sheet can be slightly modified for rendering in HTML as a table, as shown in Figure 17.37.

The Dr. Who SLU engine has a concept called *logical container* as a dialog property to be encapsulated in a semantic class. Three types of logical containers can be accessed in the definition of semantic classes. A semantic class is an AND type container if all its attributes must be evaluated successfully. If this requirement is not met, the evaluation of the AND type semantic object is considered failed, which will prompt the system to post a dialog-repair event. An OR type container requires at least one attribute to be successfully evaluated. Similarly, for an exclusive or (XOR) type container, one and only one attribute must be successfully evaluated.

```
<xsl:template match="DirectoryQuery[@not(status)]">
<TABLE border="1">
    <THEAD><TR>
        <TH>Properties</TH>
        <TH><xsl:apply-templates select="Person"/> </TH>
    </TR></THEAD>
    <TBODY><xsl:apply-templates select="DirectoryItem"/>
    </TBODY>
</TABLE>
</xsl:template>
<xsl:template match="phone">
    <TR> <TD>phone</TD> <TD> <xsl:value-of /> </TD> </TR>
</xsl:template>
```

**Figure 17.37** An HTML response-rendering rule for discourse SML of Figure 17.33. It generates a visual table representation rather than a text message [59].

Figure 17.38 shows a semantic class hierarchy corresponding to the partial plan shown in Figure 17.20. The dialog goal—to gather information for booking a flight—corresponds to the highest-level semantic class *Book Flight*. Evaluating this semantic class drives the dialog system to traverse down the semantic class structure, eventually fulfilling all the steps necessary to achieve the dialog goal. This is achieved by recursively evaluating the attributes, instantiating semantic objects actively if necessary. The logical relation of each semantic class determines the rules of instantiation and dialog repair. For instance, if the user specifies the trip to be one way only, the evaluation of the *One Way Flag* semantic class becomes successful. As the *Inbound Trip* semantic class is an XOR container, the dialog system bypasses the evaluation of the *Itinerary* attribute in the *Inbound Trip* semantic class.

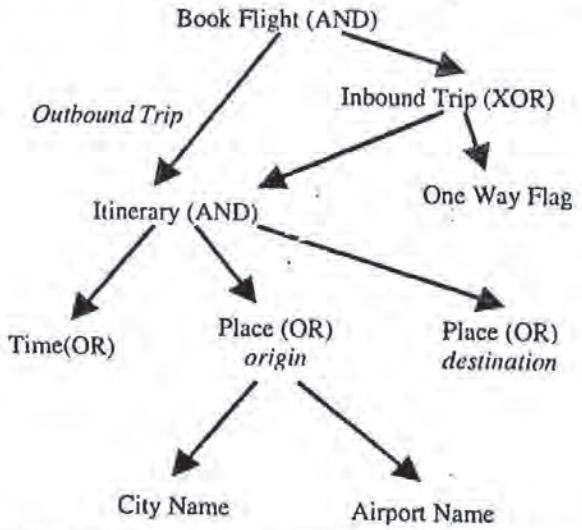



**Figure 17.38** A semantic tree hierarchy corresponding to the partial plan shown in Figure 17.20 in an airline reservation application [58].

The *Itinerary* semantic class encapsulates the basic elements to specify a one-way trip. Since it is designated as an AND type container, the dialog manager tries to acquire any missing information by actively instantiating the corresponding semantic classes it contains. The active instantiation event handlers for these classes solicit information from the user by implementing certain prompting strategy. On the other hand, the *Place* semantic class, which is used to denote both the origin and the destination, is implemented as an OR container. The user may specify the location by either the city name or the airport name.

## 17.10. HISTORICAL PERSPECTIVE AND FURTHER READING

Traditional natural language research has its roots in symbolic systems. Motivated by the desire to—understand cognitive processes, the underlying theories tend to be from linguistics and psychology. As a result, coverage of phenomena of theoretical interest (usually a rare occurrence) has traditionally been more important than developing systems with a broad coverage.

On the other hand, speech recognition research is driven to produce practical usable applications. Techniques motivated by knowledge of human processes have been less important than techniques that can be used for real applications. In recent decades, interest has grown in the use of engineering techniques in computational language processing, although the use of linguistic knowledge and techniques in engineering has lagged somewhat. The ATIS program sponsored by DARPA had a very significant influence upon the SLU research community [34]. For the first time, the research community started seriously evaluating SLU systems on a quantitative basis, which revealed that many traditional NL techniques designed for written language failed to deal with spoken language in practice.

For limited-domain SLU applications, vocabularies are typically about 2000 words. CMU's Phoenix SLU system [63] set the benchmark for domain-specific spoken language understanding in the DARPA ATIS programs. It is based on an island-driven semantic parsing approach. After years of engineering, the speech understanding error rate ranges from 6% to 41%. Since conversational repairs in human-human dialog can often be in the same range for these systems, the determining factor in these domain-specific SLU applications may not be the error rates but instead the ability of the system to manage and recover from errors. Many of these were described in detail in the *Proceedings of the DARPA Spoken Language Systems Technology Workshop* published by Morgan Kaufmann from 1991 to 1995. The special issue *of Speech Communication on Spoken Dialog* [45] also includes several state-of-the-art system descriptions.

Allen's *Natural Language Understanding* [1] is a good book on natural language understanding with a comprehensive coverage of syntactic processing, semantic processing, discourse analysis, and dialog agent. Knowledge and semantic representation comprise the most import fundamental issue for symbolic artificial intelligence. Several AI textbooks [33, 56, 65] contain comprehensive description of knowledge representation. The use of semantic frames can be traced back to case frames or structures proposed by Fillmore [16]. SAM [44] is among the first systems using semantic frames and template matcher for natural language

processing. The description of semantic classes and frames in this book mostly follows the systematic treatment of semantic classes in the Dr. Who system [58-60].

Speech-act (sometimes called dialog-act) theory was first proposed by Austin [4] and further developed by Searle [42]. It is an important concept in dialog systems. You can acquire more information about speech-act theory and its application to dialog systems from [12, 40, 43]. Cohen [10] provides a good comparison of different approaches for dialog modeling, including dialog grammar (finite state), plan-based and agent-based (dialog as teamwork). We treat agent-based dialog modeling as an extension of plan-based dialog modeling, as described in Section 17.6.2. Agent-based approach is a very popular framework for multimodal user interface, and interested readers can refer to [11]. Hudson and Newell [23] incorporate probability into finite state dialog management to handle uncertainty in input modalities, such as pen-based interface, gesture recognition, and speech recognition. J. Allen's book [1] has a systematic description of plan-based dialog systems. De Mori's *Spoken Dialogs with Computers* [39] is another excellent book that contains dialog systems and related technologies.

Much of the content in this chapter follows the architecture and implementation of semantic frame based approaches. In particular, we use plenty of descriptions and examples of the Dr. Who SLU engine developed at Microsoft Research [22, 58-60]. The description of plan-based systems is based on semantic frame representation and pattern matching. There is no need for explicit dialog-act analysis and logic reasoning, since these important knowledge sources are encapsulated in the semantic frames.

In addition to the semantic frame-based approach, there other approaches that rely on formal NL parsing, logic form representation, speech acts, and logic inference [2, 41]. Message generation for telephone application is well studied and reported in [5, 6, 49], which provide experimental results for various prompting strategies. Most evaluation schemes for the SLU systems focus on the end-to-end system. Human factors are important in overall evaluation [7, 35, 52].

## REFERENCES

[1] Allen, J., *Natural Language Understanding*, 2nd ed., 1995, Menlo Park CA, The Benjamin/Cummings Publishing Company.

[2] Allen, J.F., *et al.*, "Trains as an Embodied Natural Language System," *AAAI-95 Symposium on Embodied Language and Action*, 1995.

[3] Andernach, T., M. Poel, and E. Salomons, "Finding Classes of Dialogue Utterances with Kohonen Networks," *Proc. of the NLP Workshop of the European Conf. on Machine Learning (ECML)*, 1997, Prague, Czech Republic.

[4] Austin, J.L., *How to Do Things with Words*, 1962, Cambridge, MA, Harvard University Press.

[5] Basson, S., "Integrating Speech Recognition and Speech Synthesis in the Telephone Network," *Proc. of the Human Factors Society 36th Annual Meeting*, 1992.

[6] Basson, S., "Prompting The User in ASR Applications," *Proc. of COST232 (European Cooperation in Science and Technology) Workshop*, 1992.

[7] Basson, S., *et al.*, "User Participation and Compliance in Speech Automated Telecommunications Applications," *Proc. of the Int. Conf. on Spoken Language Processing*, 1996, pp. 1680-1683.

[8] Biber, D., *Variation Across Speech and Writing*, 1988, Cambridge University Press.

[9] Clark, H.H. and S.E. Haviland, "Comprehension and the Given-New Contract" in *Discourse production and comprehension*, R.O. Freedle, Editor 1977, Norwood, NJ, Ablex Publishing Corporation, pp. 1-38.

[10] Cohen, P., "Models of Dialogue," *Proc. of the Fourth NEC Research Symposium*, 1994, SIAM Press.

[11] Cohen, P.R., "The Role of Natural Language in a Multimodal Interface," *Proc. of the ACM Symposium on User Interface Software and Technology*, 1992, pp. 143-149.

[12] Cohen, P.R. and C.R. Perrault, "Elements of a Plan-Based Theory of Speech Acts," *Cognitive Science*, 1979, 3(3), pp. 177-212.

[13] Constantinides, P.H., S. Tchou, C. Rudnicky, "A Schema Based Approach to Dialog Control," *Proc. of the Int. Conf. on Spoken Language Processing*, 1998, pp. 409-412.

[14] Core, M. and J. Allen, "Coding Dialogs with the DAMSL Annotation Scheme," *Proc. AAAI Fall Symposium on Communicative Action in Humans and Machines*, 1997.

[15] Davies, K., *et al.*, "The IBM Conversational Telephony System For Financial Applications," *EuroSpeech'99*, 1999, Budapest, Hungary, pp. 275-278.

[16] Fillmore, C.J., "The Case for Case" in *Universals in Linguistic Theory*, E. Bach and R. Harms, eds. 1968, New York, NY, Holt, Rinehart and Winston.

[17] Galitz, W.O., *The Essential Guide to User Interface Design: An Introduction to Gui Design Principles and Techniques*, 1996, John Wiley & Sons.

[18] Grosz, B., M. Pollack, and C. Sidner, eds. *Discourse*, in Foundations of Cognitive Science, ed. M. Posner, 1989, MIT Press.

[19] Heeman, P.A., *et al.*, "Beyond Structured Dialogues: Factoring Out Grounding," *Proc. of the Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia.

[20] Hemphill, C.T., J.J. Godfrey, and G.R. Doddington, "The ATIS Spoken Language Systems Pilot Corpus," *Proc. of the Speech and Natural Language Workshop*, 1990 pp. 96-101.

[21] Hitzeman, J., *et al.*, "On the Use of Automatically Generated Discourse-Level Information in a Concept-to-Speech Synthesis System," *Proc. of the Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia, pp. 2763-2766.

[22] Huang, X., *et al.*, "MIPAD: A Next Generation PDA Prototype," *Int. Conf. on Spoken Language Processing*, 2000, Beijing, China.

[23] Hudson, S.E. and G.L. Newell, "Probabilistic State Machines: Dialog Management for Inputs with Uncertainty," *Proc. of the ACM Symposium on User Interface Software and Technology*, 1992, pp. 199-208.

[24] Hulstijn, J. and A.V. Hessen, "Utterance Generation for Trans. Dialogues," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia.

[25] Jackendoff, R.S., *X' Syntax: A Study of Phrase Structure*, 1977, Cambridge, MA, MIT Press.

[26] Jelinek, F., *et al.*, "Decision Tree Parsing using Hidden Derivational Model," *Proc. of the ARPA Human Language Technology Workshop*, 1994, pp. 272-277.

[27] Jurafsky, D., L. Shriberg, and D. Biasca, *Switchboard SWBD-DAMSL Shallow-Discourse-Function Annotation Coders Manual, Draft 13*, 1997, http://www.colorado.edu/linguistics/jurafsky/manual.august1.html.

[28] LDC, *Linguistic Data Consortium*, 2000, http://www.ldc.upenn.edu/ldc/noframe.html.

[29] Miller, S., *et al.*, "Recent Progress in Hidden Understanding Models," *Proc. of the ARPA Spoken Language Systems Technology Workshop*, 1995, Austin, Texas, Morgan Kaufmann, Los Altos, CA, pp. 22-25.

[30] Miller, S. and R. Bobrow, "Statistical Language Processing Using Hidden Understanding Models," *Proc. of the Spoken Language Technology Workshop*, 1994, Plainsboro, New Jersey, Morgan Kaufmann, Los Altos, CA, pp. 48-52.

[31] Minsky, M., "A Framework for Representing Knowledge" in *The Psychology for Computer Vision*, P.H. Winston, Editor 1975, New York, NY, McGraw-Hill.

[32] Nilsson, N.J., *Principles of Artificial Intelligence*, 1982, Berlin, Germany, Springer-Verlag.

[33] Nilsson, N.J., *Artificial Intelligence: A New Synthesis*, 1998, Academic Press/Morgan Kaufmann.

[34] Pallett, D.S., *et al.*, "1994 Benchmark Tests for the ARPA Spoken Language Program," *Proc. of the 1995 ARPA Human Language Technology Workshop*, 1995, pp. 5-36.

[35] Polifroni, J., *et al.*, "Evaluation Methodology for a Telephone-Based Conversational System," *The First Int. Conf. on Language Resources and Evaluation*, 1998, Granada, Spain, pp. 42-50.

[36] Prevost, S. and M. Steedman, "Specifying Intonation from Context for Speech Synthesis," *Speech Communication*, 1994, **15**, pp. 139-153.

[37] Reinhart, T., *Anaphora and Semantic Interpretation*, Croom Helm Linguistics Series, 1983, University of Chicago Press.

[38] Roberts, D., *The Existential Graphs of Charles S. Peirce*, 1973, Mouton and Co.

[39] Sadek, D. and R. De Mori, "Dialogue Systems" in *Spoken Dialogues with Computers*, R. De Mori, Editor 1998, London, UK, pp. 523-561, Academic Press.

[40] Sadek, M.D., "Dialogue Acts are Rational Plans," *Proc. of the ESCA/ETRW Workshop on the Structure of Multimodal Dialogue*, 1991, Maratea, Italy, pp. 1-29.

[41] Sadek, M.D., *et al.*, "Effective Human-Computer Cooperative Spoken Dialogue: The AGS Demonstrator," *Proc. of the Int. Conf. on Spoken Language Processing*, 1996, Philadelphia, Pennsylvania, pp. 546-549.

[42] Searle, J.R., *Speech Acts: An Essay in the Philosophy of Language*, 1969, UK, Cambridge University Press.

[43] Searle, J.R. and D. Vanderveken, *Foundations of Illocutionary Logic*, 1985, Cambridge University Press.

[44] Shank, R., *Conceptual Information Processing*, 1975, North Holland, Amsterdam, The Netherlands.

[45] Shirai, K. and S. Furui, "Special Issue on Spoken Dialogue," *Speech Communication*, 1994, **15**.

[46] Shriberg, E.E., R. Bates, and A. Stolcke, "A Prosody-Only Decision-tree Model for Disfluency Detection," *Proc. Eurospeech*, 1997, Rhodes, Greece, pp. 2383-2386.

[47] Sidner, C., "Focusing in the Comprehension of Definite Anaphora" in *Computational Model of Discourse*, M. Brady, Berwick, R., eds., 1983, Cambridge, MA, pp. 267-330, The MIT Press.

[48] Sidney, S. and N.J. Castellan, *Nonparametric Statistics for the Behavioral Sciences*, 1988, McGraw Hill.

[49] Sorin, C. and R.D. Mori, "Sentence Generation" in *Spoken Dialogues with Computers*, R.D. Mori, Editor 1998, London, UK, Academic Press, pp. 563-582.

[50] Souvignier, B., *et al.*, "The Thoughtful Elephant: Strategies for Spoken Dialog Systems," *IEEE Trans. on Speech and Audio Processing*, 2000, **8**(1), pp. 51-62.

[51] Sowa, J.F., *Knowledge Representation: Logical, Philosophical, and Computational Foundations*, 1999, Brooks Cole Publishing Co.

[52] Springer, S., S. Basson, and J. Spitz, "Identification of Principal Ergonomic Requirements for Interactive Spoken Language Systems," *Int. Conf. on Spoken Language Processing*, 1992, pp. 1395-1398.

[53] Standards, N.C.f.I.T., *Conceptual Graph Standard Information*, 1999, http://www.bestweb.net/~sowa/cg/cgdpansw.htm.

[54] Steedman, M., ed. *Parsing Spoken Language Using Combinatory Grammars*, in Current Issues in Parsing Technology, ed. M. Tomita, 1991, Kluwer Academic Publishers.

[55] Steedman, M., "Information Structure and the Syntax-Phonology Interface," *Linguistic Inquiry*, 2000.

[56] Tanimoto, S.L., *The Elements of Artificial Intelligence: An Introduction Using Lisp*, 1987, Computer Science Press, Inc.

[57] Walker, M., *et al.*, "PARADISE: A Framework for Evaluating Spoken Dialogue Agents.," *Proc. of the 35th Annual Meeting of the Association for Computational Linguistics (ACL-97)*, 1997, pp. 271-280.

[58] Wang, K., "An Event Driven Model for Dialogue Systems," *Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia pp. 393-396.

[59] Wang, K., "Implementation of Dr. Who Dialog System Using Extended Markup Languages," *Int. Conf. on Spoken Language Processing*, 2000, Beijing, China.

[60] Wang, K., "A Plan-Based Dialog System With Probabilistic Inferences," *ICSLP*, 2000, Beijing, China.

[61] Wang, Y., "A Robust Parser For Spoken Language Understanding," *Eurospeech*, 1999, Budapest, Hungary, pp. 2055-2058.

[62] Wang, Y., M. Mahajan, and X. Huang, "A Unified Context-Free Grammar and N-Gram Model for Spoken Language Processing," *Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, Turkey, pp. 1639-1642.

[63] Ward, W., "Understanding Spontaneous Speech: The Phoenix System," *Proc. Int. Conf. on Acoustics, Speech and Signal Processing*, 1991, Toronto, Canada, pp. 365-367.

[64] Ward, W. and S. Issar, "The CMU ATIS System," *Proc. of the ARPA Spoken Language Systems Technology Workshop*, 1995, Austin, Texas, Morgan Kaufmann, Palo Alto, CA.

[65] Winston, P.H., *Artificial Intelligence*, 3rd ed, 1992, Reading, MA, Addison-Wesley.

[66] Young, S.R., *et al.*, "High Level Knowledge Sources in Usable Speech Recognition Systems," *Communications of the Association for Computing Machines*, 1989, **32**(2), pp. 183-194.

# CHAPTER 18

# Applications and User Interfaces

The ultimate impact of spoken language technologies depends on whether you can fully integrate the enabling technologies with applications so that users find it easy to communicate with computers. How to effectively integrate speech into applications often depends on the nature of the user interface and application. This is why we group user interface and application together in this chapter. In discussing some general principles and guidelines in developing spoken language applications, we must look closely at designing the user interface.

A well-designed user interface entails carefully considering the particular user group of the application and delivering an application that works effectively and efficiently. As a general guideline, you need to make sure that the interface matches the way users want to accomplish a task. You also need to use the most appropriate modality at the appropriate time to assist users to achieve their goals. One unique challenge in spoken language applications is that neither speech recognition nor understanding is perfect. In addition, the spoken command can be ambiguous, so the dialog strategy described in Chapter 17 is necessary to clarify the goal of the speaker. There are always mistakes you have to deal with. It is critical

that applications employ necessary interactive error-handling techniques to minimize the impact of these errors. Application developers should therefore fully understand the strengths and weaknesses of the underlying speech technologies and identify the appropriate place to use the spoken language technology effectively.

This chapter mirrors Chapter 2, in the sense that you need to incorporate all the needed components of speech communication to make a spoken language system work well. It is important also to have your applications developed based on some standard application programming interfaces (API), which ensures that multiple applications work well with a wide range of speech components provided by different speech technology providers.

## 18.1. APPLICATION ARCHITECTURE

A typical spoken language application has three key components. It needs an engine that can be either a speech recognizer or a spoken language understanding system. An *application programming interface* (API) is often used to facilitate the communication between the engine and application, as illustrated in Figure 18.1. Multiple applications can interact with a shared speech engine via the speech API. The speech engine may be a CSR engine, a TTS engine, or an SLU engine. The interface between the application and the engine can be distributed. For example, you can have a client-server model in which the engine is running remotely on the server.

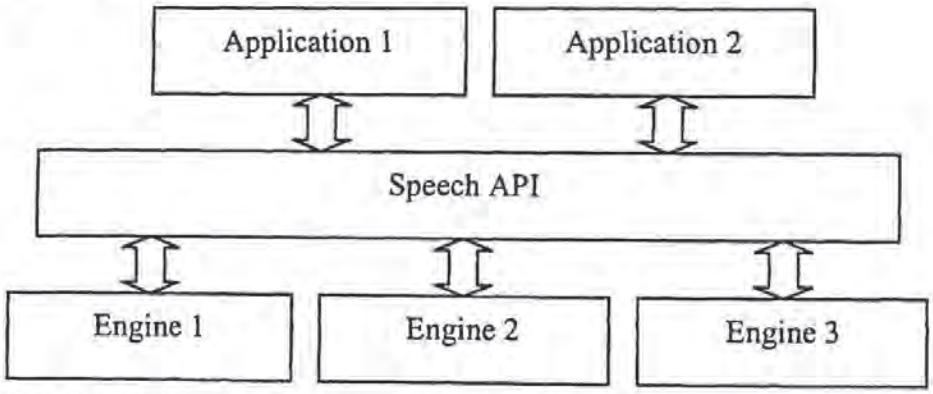



**Figure 18.1** In a typical spoken language application architecture, multiple applications can interact with a shared speech engine via the speech API. The speech engine may be a speech recognizer, a TTS converter, or an SLU engine.

For a given API, there is typically an associated toolkit that provides a good development environment and the tools you need in order to build speech applications. You don't need to understand the underlying speech technologies to fully take advantage of state-of-the-art speech engines. Industry-standard based applications can draw upon support from many different speech engine vendors, thus significantly minimizing the cost of your applications development. For the widely used Microsoft Windows®, Microsoft's speech API

(SAPI) brings both engine and application developers together.[1] Alternative standards are available, such as VoiceXML[2] and JSAPI[3].

## 18.2. TYPICAL APPLICATIONS

There are three broad classes of applications that require different UI design:

- *Office:* This includes the widely used desktop applications such as Microsoft Windows and Office.
- *Home*: TV and kitchen are the centers for home applications. Since home appliances and TV don't have a keyboard or mouse, the traditional GUI application can't be directly extended for this category.
- *Mobile*: Cell phone and car are the two most important mobile scenarios. Because of the physical size and the hands-busy and eyes-busy constraints, the traditional GUI application interaction model requires significant modification.

This section provides descriptions of typical spoken language applications in these three broad classes. Spoken language has the potential to provide a consistent and unified interaction model across these three classes, albeit for these different application scenarios you still need to apply different user interface design principles.

### 18.2.1. Computer Command and Control

One of the earliest prototypes for speech recognition is *command and control*, which is mainly used to navigate through operating system interfaces and applications running under them. For example, Microsoft Agent is a set of software services that supports the presentation of software agents as interactive personalities within the Microsoft Windows or the Internet Explorer interface. Its command-and-control speech interface is an extension and enhancement of the existing interactive modalities of the Windows interface. It has a character called Peedy, shown in Figure 18.2, which recognizes your speech and talks back to you using a Microsoft SAPI compliant command-and-control speech recognizer and text-to-speech synthesizer.

The speech recognizer used in these command-and-control systems is typically based on a context-free grammar (CFG) decoder. Either developers or users can define these grammars. Associated with each legal path in the grammar is a corresponding executable event that can map a user's command into appropriate control actions the user may want. They possess a built-in vocabulary for the menus and other components. The vocabulary can also be dynamically provided by the application. Command-and-control speech recognition allows the user to speak a word, phrase, or sentence from a list of phrases that the computer is expecting to hear. The number of different commands a user might speak at any time can

[1] http://www.microsoft.com/speech
[2] http://www.voicexml.org/
[3] http://java.sun.com/products/java-media/speech/

be in the hundreds. Furthermore, the commands are not just limited to fixed ones but can also contain other open fields, such as "*Send mail to <Name>*" or "*Call <digits>*". With all of the possibilities, the user is able to speak thousands of different commands. As discussed in Chapter 17, a CFG-based recognizer is often very rigid, since it may reject the input utterance that contains a sentence slightly different from what the CFG defines, leading to an unfriendly user experience.




**Figure 18.2** A talking character *Peedy*[4] as used in Microsoft Agent. Reprinted with permission from Microsoft Corporation.

Command-and-control recognition might be useful in some of the following situations:

- *Answering questions*. An application can easily be designed to accept voice responses to message boxes and wizard screens. Most speech recognition engines can easily identify *Yes*, *No*, and a few other short responses.
- *Accessing large lists*. In general, it's faster for a user to speak one of the names on a list, such as "*Start running calculator,*" than to scroll through the list to find it. It assumes that the user knows what is in the list. Laurila and Haavisto [23] summarized their usability study of inexperienced users on name dialing. Although the study is based on the telephone handset, it has a similar implication for computer desktop applications.
- *Activating macros*. Speech recognition lets a user speak a more natural word or phrase to activate a macro. For example, "*Spell check the second paragraph*" is easier for most users to remember than the CTRL+F5 key combination after selecting the second paragraph. But again, the user must know the command. This is where most simple speech applications fail. The competition is not CTRL+F5 itself, it is the memory of most users.
- *Facilitating dialog between the user and the computer*. As discussed in Chapter 17, speech recognition works well in situations where the computer essen-

[4] *Peedy* © 1993-1998 Microsoft Corporation.

tially asks the user: "*What do you want to do*?" and branches according to the reply (somewhat like a wizard). For example, the user might reply, "*I want to book a flight from New York to Boston.*" After the computer analyzes the reply, it clarifies any ambiguous words (*Did you say New York*?). Finally, the computer asks for any information that the user did not supply, such as "*At what day and time do you want to leave*?"

- *Providing hands-free computing.* Speech recognition is an essential component of any application that requires hands-free operation; it also can provide an alternative to the keyboard for users who are unable to or prefer not to use one. Users with repetitive-stress injuries or those who cannot type may use speech recognition as the sole means of controlling the computer. As discussed in later sections, hands-free computing is important for accessibility and mobility.
- *Humanizing the computer.* Speech recognition can make the computer seem more like a person—that is, like someone whom the user talks to and who speaks back. This capability can make games more realistic and make educational or entertainment applications friendlier.

The specific use of command and control depends on the application. Here are some sample ideas and their uses:

- *Games and entertainment:* Software games are some of the early adopters of command-and-control speech recognition. One of the most compelling uses of speech recognition technology is in interactive verbal exchanges and conversation with the computer. With games such as flight simulators, for example, traditional computer-based characters can now evolve into characters the user can actually talk to. While speech recognition enhances the realism and fun in many computer games, it also provides a useful alternative to game control. Voice commands provide new freedom for the user.
- *Document editing*: Command and control is useful for document editing when you wish to keep your hands on the keyboard to type, or on the mouse to drag and select. This is especially true when you have to do a lot of editing that requires you to move to menus frequently. You can simultaneously speak commands for manipulating the data that you are working on. A word processor might provide commands like "*bold, italic*" and "*change font.*" A paint package might have "*select eraser*" or "*choose a wider brush.*" Of course, there are users who won't prefer speaking a command to using keyboard equivalents, as they have been using the latter for so long that the combinations have become for them a routine part of program control. But for many people, keyboard equivalents are a lot of hard-to-remember shortcuts. Voice commands provide these users with the means to execute a command directly.

For most of the existing applications, before an application starts a command-and-control recognizer, it must first give the recognizer a *list* of commands to listen for. The list might include commands like "*minimize window*," "*make the font bold*," "*call extension <digit> <digit> <digit>*," or "*send mail to <name>*." If the user speaks the command as it is designed, he/she typically gets very good accuracy. However, if the user speaks the command differently, the system typically either does not recognize anything or erroneously recognizes something completely different. Applications can work around this problem by:

- Making sure the command names are intuitive to users. For many operations like minimizing a window, nine out of ten users will say *minimize* window without prompting.
- Showing the command on the screen. Sometimes an application displays a list of commands on the screen. Users naturally speak the same text they see.
- Using word spotting as discussed in Chapter 9. Many speech recognizers can be told to just listen for one keyword, like *mail*. This way the user can speak, "*Send mail,*" or "*Mail a letter,*" and the recognizer will get it. Of course, the user might say, "*I don't want to send any mail,*" and the computer will still end up sending mail.
- Employing spoken language understanding components as discussed in Chapter 17.
- Employing user studies to collect data on frequently spoken variations on commands so that the coverage is enhanced.

### 18.2.2. Telephony Applications

Speech is the only available modality for telephony applications besides the awkward-to-use DTMF interface. The earliest uses of speech technology in business were *interactive voice response* (IVR) systems. These systems include *infoline* services in the ad-supported local newspapers, offering everything from world news to school homework assignments at the touch of a few buttons. So what's the big deal with a speech telephony application? It offers greater breadth, ease of use, and interactivity. Navigating by voice rather than by keypad offers more options and quicker navigation. It also works better while you're driving.

To make a successful IVR application, you need to have speech input, output, and related dialog control. People have used IVR systems over the telephone to navigate the application based on the menu option to provide digit strings, such as the credit card numbers, to the application. Such system typically has a small to medium vocabulary. Today, you can use IVR to get stock quotes, people's telephone number, and other directory-related information. For example, you can call AT&T universal card services and the application asks you to speak your 16-digit card number. Most of these IVR systems use recorded messages instead of synthetic speech because the quality of TTS is still far from humanlike. Since speech output is a slow method to present information, it is important to be as brief as possible. Reducing the presentation of repetitive data can shorten the speech output significantly.

Voice portals that let you talk your way to Web-based information from any phone are one class of compelling telephony applications. Linked to specially formatted Web sites and databases, the portals deliver what amounts to customized real-time news radio. You can tailor voice portals much as you do Web portals like *Yahoo!*[3], AOL[4], or MSN[5]. But surfing is restricted to the very limited subsets of information the portals choose to offer. These services typically avoid using synthesized speech. For options like news updates they rely on sound bites recorded by announcers. There are a number of free voice portals available, including TellMe, BeVocal, HeyAnita, Quack.com. Table 18.1 illustrates some of their features.

**Table 18.1** Some free voice portal features. These portals are being developed and will roll out more features.

| Category | Audiopoint[5] | Tell Me[6] |
|---|---|---|
| Traffic | Yes | Yes |
| Weather | U.S. and world cities | U.S. |
| News | Yes | Yes |
| Financial | Yes | Yes |
| Sports | Yes | Yes |
| Airline info | No | Yes |
| Restaurants | No | U.S. |
| Entertainment | Yes | Yes |
| Personalization | Yes | Yes |

Digital wireless telephony applications could make full use of a client-server architecture because of limited computing resources of the client. The server performs most of the needed processing. The client can either send the speech waveform (as used in standard telephone) or the spectral parameters such as MFCC coefficients. Using a quantized MFCC (see Chapter 6) at 4.5 kbps, no loss of accuracy can be achieved [18]. The Aurora project tries to standardize the client server communication protocol based on the quantized MFCC coefficients[7] [6].

When people are engaged in a conversation, even if they have the graphical interface in front of them, they seldom use the vocabulary from the interface (unless prompted by TTS or the speaker). This has an important implication for the UI design. The use of a discourse segment pop cue such as *"What now?"* or *"Do you want to check messages?"* could reorient users (especially after a subdialog) and help them figure out what to say next. Wildfire[8] uses such pop cues extensively. The right feedback is essential, because speech recognition is not perfect. Designers should verify only those commands that might destroy data or trigger future events. People become frustrated very quickly if the error feedback is re-

[3] http://www.myaudiopoint.com or call 1 -888-38-AUDIO.
[4] http://www.tellme.com or call 1-800-555-TELL.
[5] http://www.etsi.org/stq
[6] http://www.wildfire.com/

petitive. It would be nice to design the error messages with different prompts according to recognition confidence measures and the dialog context.

Most current telephony systems do not have much intelligent dialog management beyond a simple state-based dialog model. Advanced dialog management as discussed in Chapter 17 is critical for error repairs, anaphora resolution, and context management. There are a number of institutions pursuing advanced telephony spoken language interfaces. You can find some of these excellent programs at Carnegie Mellon University[9] and Massachusetts Institute of Technology.[10]

### 18.2.3. Dictation

Dictation is attractive to many people, since speaking is generally faster than typing, especially in East Asian languages such as Chinese. There are a number of general-purpose dictation software products such as IBM's ViaVoice®, Dragon's NaturallySpeaking®, Lernout&Hauspie's Voice Xpress®, as well as the dictation capability built into Microsoft Office 2001. There are also vertical markets for dictation applications. Radiologists and lawyers are two examples of such niche markets. They are generally pressed for time. Radiologists are often hands- and eyes-busy when at work. Lawyers' work is often language intensive and they need to write long documents and reports. Both radiologists and lawyers have been using dictation devices, and they have a strong need to have customized language models for their specialized vocabulary. In fact, because of the constraints of the language they use, the perplexity is often much smaller than in the general business articles, leading to improved performance for both the medical and legal segments.

In dictation applications, it is important to convert speech into the text you would like to see in its written form. This implies that punctuation should be added automatically, dates and times should be converted into the form that is conventionally used, and appropriate capitalization and homonym disambiguation should all be seamless to users. Dictation is typically associated with word processing applications, which provide for a nice separation of input modes based upon the division of the primary task into subactivities:

- Text entry
- Command execution, such as *cross-out* to delete the previous word, or *capitalize-that*
- Direct manipulation activities such as cursor positioning and text selection

It is likely that separate single input modes for each activity can increase efficiency. In a study conducted by Morrison et al. [28], subjects switched modality from speech input to typed input while issuing text-editing commands and found the switch of modality *disruptive*. Karat et al. [19] compared the state-of-the-art dictation product with keyboard input.

[9] http://www.speech.cs.cmu.edu/speech/
[10] http://www.sls.lcs.mit.edu/sls/

They reported that several of the initial-use subjects commented that keyboard entry seemed *much more natural*. This reflects the degree to which some people have become used to using keyboards and the fact that speech recognition still makes too many mistakes. Experienced users can enter transcription text at an average rate of 107 uncorrected words per minute. However, correction took them over three times as long as entry time on average, leading to an average input rate nearly twice as slow as keyboard (including correction). Thus, unless the accuracy of speech recognition and its error correction can be significantly improved for the mass market, text entry is best performed by keyboard.

The mouse is generally accepted as being well suited to direct manipulation activities. Further, based on human-factors studies, full typed input of commands, single keyboard presses, and accelerator keys is not as efficient as speech-activated commands [26, 33]. The case for the utility of speech input in word-processing applications relies upon its superiority over the mouse with respect to the activation of commands. Word-processing and text-entry applications are also naturally *hands-busy, eyes-busy* applications. It is inconvenient and time consuming for the user to have to interrupt the typing of text or to move his/her eyes from the work in order to execute word-processing commands. This is not necessary with speech activation of commands.

To improve dictation throughput, you should make it as easy as possible for users to correct mistakes. As illustrated in Figure 18.3, you can provide easy access to the *Correction Window* so the user can correct mistakes that the recognizer made. Thus the quality of *n*-best or word-lattice decoding discussed in Chapter 13 is critical. We use examples drawn from Microsoft Dictation[11] to illustrate these concepts.

You should also allow the user to train the system so it can adapt to different accent and environment conditions. A typical training wizard, illustrated in Figure 18.4, asks users to read a number of sentences. It is important to use a confidence measure, as discussed in Chapter 9, to filter out unwanted training data so that you can eliminate possible divergences of training the acoustic model. In Figure 18.4, the user is prompted with the sentence shown in the window. As the user reads the sentence, the black progress bar covers the corresponding word. If the user misreads the word, the progress bar typically stays where it is.

Another practical problem for most dictation products is that they have about 60,000 to 100,000 words in the vocabulary, which contains all the inflected forms, so *work* and *works* are considered as two different words. As the vocabulary is limited (for detailed discussion on vocabulary selection, see Chapter 11), you should provide a capability to dynamically add new words to the system. This capability is illustrated in Figure 18.5. You can listen to the new word you added to see if the computer guessed the pronunciation of the word correctly or not. If you want to pronounce the word differently, you can either type in the words that sound the way you want it to or type in the phonetic sequence.

[11] Microsoft Dictation was developed by the authors at Microsoft Research, and was included for free in the SDK4.0.

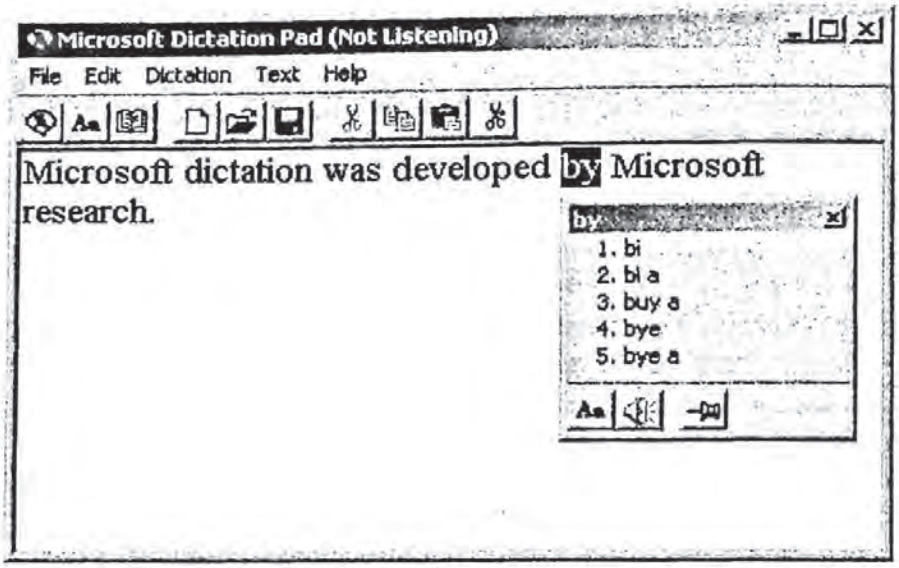



**Figure 18.3** *Correction window* in Microsoft Dictation. By clicking the dictated word, the alternative words are listed in the correction window. If the dictated word is wrong, you can click the word in the correction window to easily replace the misrecognized word. Reprinted with permission from Microsoft Corporation.

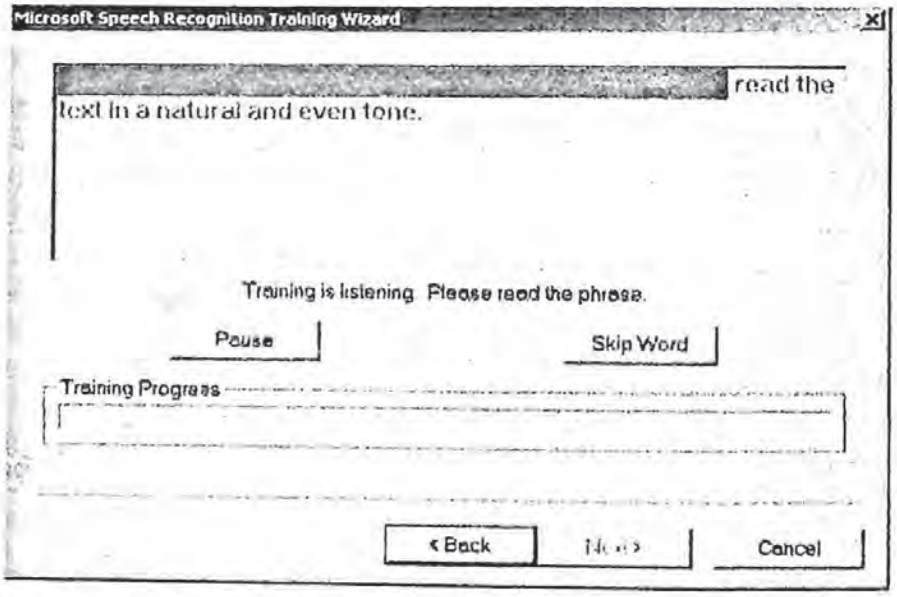



**Figure 18.4** Training wizard in Microsoft Dictation. Reprinted with permission from Microsoft Corporation.

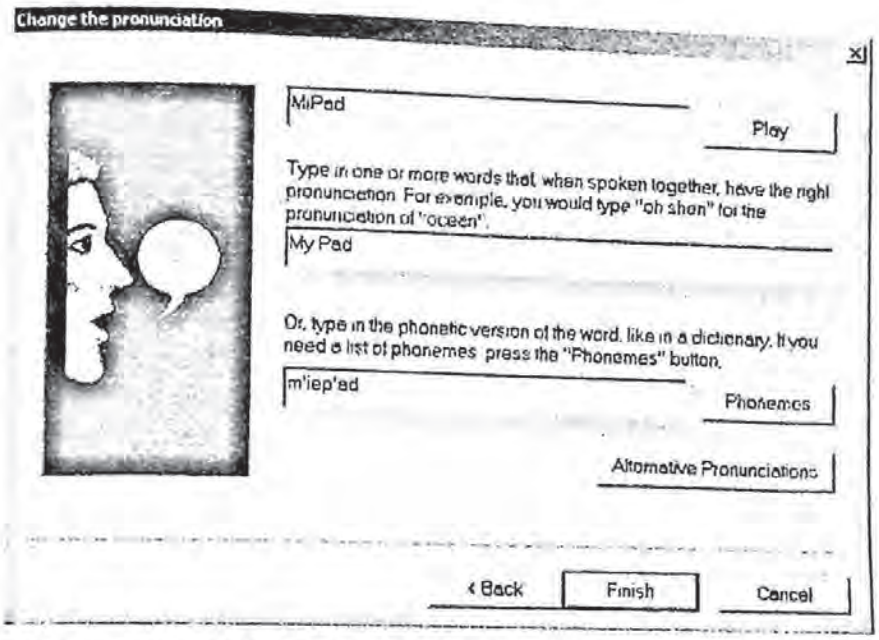



**Figure 18.5** New word addition in Microsoft Dictation. Reprinted with permission from Microsoft Corporation.

## 18.2.4. Accessibility

Approximately 10–15 percent of the population has an impairment that affects their ability to use computers efficiently. Accessibility is about designing applications that everyone can use. The GUI actually makes accessibility a more critical issue. People with disabilities face more usability issues than those without. There are a number of Web sites dedicated to accessibility.[12]

People who cannot effectively use a keyboard or a mouse can use speech commands to control the computer and use the dictation software to input text. For example, Dragon's dictation software lets you control everything by voice, making it extremely useful for people with repetitive stress injury (RSI). A screen reader is a program that uses TTS to help people who have reading problems or are vision impaired. Windows® 2000 includes one such screen reader, called Narrator, which reads what is displayed on the screen: the contents of the active window, menu options, and text that the user types. Users can customize the way these screen elements are read. They can also configure Narrator to make the mouse pointer follow the active item on the screen and can adjust the speed, volume, and pitch of the narrator voice.

[12] http://www.el.net/CAT/index.html (The Center for Accessible Technology).
http://www.lighthouse.org/print_leg.htm (Lighthouse International).
http://www.w3c.org/wai (World Wide Web Consortium's Accessibility Page).
http://www.microsoft.com/enable (Microsoft Accessibility Home Page).

### 18.2.5. Handheld Devices

For handheld devices such as *smart phones* or *Personal Digital Assistants* (PDA), speech, pen, and display can be used effectively to improve usability. The small form-factor implies that most of the functions have to be hidden. Speech is particularly suitable to access information that can't be seen on the display. Current input methods are generally pen- or soft keyboard-based. They are slower than dictation-based text input. For form filling, speech is also particularly suitable to combine multiple parameters in a single phrase. The increasing popularity of cell phones is likely to be the major thrust for using handheld devices to access information.

Potential usability problems of handheld devices are the lack of privacy and the problem of environmental noise. It is hard to imagine that we can have 100 people in a conference room talking to their handheld devices simultaneously. Since these devices are also often used in a noisy environment, you need to have a very robust speech recognition system to deal with signals around 15 dB. In Section 18.5 we illustrate the process of designing and developing an effective speech interface and application for a handheld device.

### 18.2.6. Automobile Applications

People already use cell phones extensively when driving. Safety is an important issue whenever a driver takes his/her eyes off the road or his/her hands off the steering wheel. In many countries it is illegal to use a cellular phone while driving. Speech-based interaction is particularly suitable for this eyes-busy and hands-busy environment. However, it is a challenge to use speech recognition in the car due to the relative high noise environment. Environment robustness algorithms, discussed in Chapter 10, are necessary to make automobile applications compelling.



**Figure 18.6** Clarion in-dash AutoPC.

One such system is Clarion's AutoPC, shown in Figure 18.6. It enables drivers to access information by speech while driving. You can tell your car what to do by speech. AutoPC has a vocabulary of more than 1200 words. It understands commands to change radio stations, provide turn-by-turn directions based on its Global Positioning System (GPS) navigation unit, or read e-mail subject lines using TTS. Running on Microsoft's Windows CE operating system, AutoPC's control module fits in a single dashboard slot, with the computer and six-disc CD changer stored in the trunk.

### 18.2.7. Speaker Recognition

The HMM-based speech recognition algorithms discussed in earlier chapters can be modified for speaker recognition, where a speaker model is needed for each speaker to be recognized. The term speaker recognition is an umbrella that includes speaker identification, speaker detection, and speaker verification. Speaker identification consists in deciding who the person is, what group the person is a member of, or that the person is unknown. Speaker detection consists in detecting if there is a speaker change. This is often needed in speech recognition, as a better speaker model can be used for improved accuracy. Speaker verification consists in verifying a person's claimed identify from his/her voice, which is also called speaker authentication, talker verification, or voice authentication.

In text-dependent approaches, the system knows the phrase, which can be either fixed or prompted to the user. The speaker speaks the required phrase to be identified or verified. The speaker typically has to enroll in the system by presenting a small number of speech samples to build the corresponding speaker model. Since the amount of training data is small, the parameter tying techniques discussed in Chapter 9 are very important.

Speaker recognition is a performance biometric, i.e., you perform a task to be recognized. It can be made fairly robust against noise and channel variations, ordinary human changes, and mimicry of humans and tape recorders. Campbell [3] and Furui [9] provided two excellent tutorial papers on recent advances in speaker recognition.

In general, accuracy is much higher for speaker recognition than for speech recognition. The best speaker recognition systems typically use the same algorithms, such as the ways to deal with environment noises and hidden Markov modeling, found in speech recognition systems. Applicable speaker recognition services include voice dialing, banking over a telephone network, telephone shopping, database access services, information and reservation services, forensic application, and voice login. There are a number of commercial speaker verification systems, including Apple's voice login for its iMac[*] and Sprint's Voice FONCARD[*].

## 18.3. SPEECH INTERFACE DESIGN

No single unified interaction theory has emerged so far to guide user interface design. Study of the interaction has been largely associated with developing practical applications, having its roots in human factors and user studies. Practitioners have postulated general principles that we discuss in this section.

### 18.3.1. General Principles

There are a number of books on graphical user interface (GUI) design [10, 11, 17, 29, 41]. Usability research also generated a large body of data that is useful to UI designers. The Human Factors and Ergonomic Society (HFES) is developing the HFES-200 standard—

*Ergonomics of Software User Interface*, which has many insightful implications for interface design. The most important criteria, in order of importance, are:

- Effectiveness or usefulness
- Efficiency or usability
- User satisfaction or desirability

These three general points mean that, first and foremost, a user interface must work, so the user can get the job done. After that, it is important that the user interface accomplishes its task as productively as possible. Last, but not least, the user interface can be further improved by focusing on user satisfaction as measured by means of questionnaires and user feedback.

The major difference between GUI and spoken language interface is that speech recognition or understanding can never be perfect. Imagine you are designing for a GUI and your input method is a keyboard and a mouse where the error rate is about 5% for the key or mouse presses. These 5% errors make it much more difficult to have an effective user interface. In GUI applications, when you mistype or misclick, you regard these errors as your own mistakes. In spoken language applications, all the misrecognition errors are regarded as system errors. This psychological effort is a very important factor in the UI design. Thus it is not stretching to say that most of the design work for spoken language applications should be on how to effectively manage what to do when something goes wrong. This should be the most important design principle to remember.

We also want to stress the importance of iteratively testing the UI design. In practice, users, their tasks, and the work environment are often too varied to be able to have a good design without going through a detailed and thorough test program, especially for new areas like spoken language applications.

#### 18.3.1.1. Human Limitations

Applications make constant demands on human capabilities. When we design the user interface, we must understand that humans have cognitive, auditory, visual, tactile, and motor limits. As a general principle, we should make the user interface easy to use, and not overload any of these limits.

For example, if the font is too small, the speech volume is too low, or the application requires people to constantly switch between a mouse and a keyboard, we have a poor interface, leading to lowered productivity. People have trouble remembering long complex messages spoken to them in a continuous fashion. If these messages are delivered with synthesized speech, the problem is worsened, because of the greater attention required to understand the synthesized speech. Effective user interface design requires a balance of easy access to information and an awareness of human limitations in processing this information. In general, people tend to handle well 7 (+ or – 2), pieces of information at a time.

### 18.3.1.2. User Accommodation

The interface and application should accommodate users as much as possible. This means that the design needs to match the way users want to do their work, not the other way around. People always have, or quickly create, a mental model of how the interface operates. You need to design and communicate a conceptual model that works well with the users' mental model. Their mental model should match the actual interface and the task, which is very important for the designer to consider consciously.

To accommodate users' needs, you must ask whether the interface you are designing helps people get their work done efficiently. Will it be easy to learn and use? Will it have an attractive and appropriate design? Will the interface fit individual tasks with a suitable modality? Will the interface allow the user to perceive that they are in control? Will the interface be flexible enough to allow the user to adjust the design for custom use? Does the interface have an appropriate tempo? Notice that speech is temporal. Does the interface contain sufficient user support if the user needs or requests it? Does the interface respond to the use in a timely fashion? Does the interface provide appropriate feedback to the user about the results of the actions and application status?

### 18.3.1.3. Modes of Interaction

The interface is unimodal if only one mode is used as both input and output modality. By contrast, multimodal systems use additional modalities in exchanging information with their users. When we interact with each other, we often rely on multimodal interaction, which is normal for us.

When speech is combined with other modalities of information presentation and exchange, it is generally more effective than a unimodal interface, especially for dealing with errors from speech recognition and understanding.

As an input modality, speech is generally not as precise as mouse or pen to perform position-related operations. Speech interaction can also be adversely affected by the ambient noise. When privacy is of concern, speech is also disadvantageous, since others can overhear the conversation. Despite these disadvantages, speech communication is not only natural but also provides a powerful complementary modality to enhance the existing keyboard- and mouse-based graphical user interface. As an input-output modality, speech has a number of unique characteristics in comparison with the traditional modalities:

- Speech does not require any screen real estate in the interface. For ubiquitous computing, this is particularly suitable for devices that are as small as our watches.
- Speech can be used at a distance, which makes it ideal for hands-busy and eyes-busy situations such as driving.

- Speech is expressive and very suitable to bring information that is hidden to the forefront. This is particularly suitable in the GUI environment or with handheld devices as the monitor space is always limited.
- Speech is omnidirectional and can communicate information to multiple users easily. However, this also has implications relating to privacy.
- Speech is temporal and sequential. Once uttered, auditory information is no longer available. This can place extra memory burden on the user and severely limit the ability to scan, review, and cross-reference information.
- For social interfaces, speech plays a very important role. For example, if we need to pose a question, make a remark, and delegate a task, there is no more natural way than speech itself.

Because of these unique features, we need to leverage the strengths and overcome the technology limitations that are associated with the speech modality. The conventional GUI relies on the visual display of objects of interest, the selection by pointing, and continuous feedback. Direct manipulation is generally inadequate for supporting fundamental transactions in applications such as word processing and database queries, as there are only limited means to identify objects, not to mention that ambiguity in the meanings of icons and limitations in screen display space further complicate the usability of the interface.

Generally, multimodality can dramatically enhance the usability of speech, because GUI and speech have complementary strengths and weaknesses. The strengths of one can be used to offset the weaknesses of others. For example, pen and speech can be complementary and they can be used very effectively for handheld devices. You can use *Tap and Talk* as discussed in Section 18.4 to activate microphone and select appropriate context for speech recognition. The respective strengths and weaknesses are summarized in Table 18.2. The advantage of pen is typically the weakness of speech and vice versa. This implies that user interface performance and acceptance could increase by combining both. Thus, visible, limited, and simple actions can be enhanced by nonvisible, unlimited, and complex actions. The multimodal interface is particularly suitable to people for completing contrastive functions or when task attributes are perceived as separable. For example, speech and pen can be used in different ways to designate a shift in context or functionality. Although most people may not prefer to use speech and pen simultaneously, they like to use speech to enter data and pen for corrections. Other contrastive tasks may include data versus commands and digits versus text. When inputs are perceived as integral, unimodal interfaces are efficient and work best.

**Table 18.2** Complementary strengths of pen and speech for multimodal user interface.

| Pen | Speech |
|---|---|
| Direct manipulation, requires hands and eyes | Hands/eyes-free manipulation |
| Good at performing simple actions | Good at performing complex actions |
| Visual feedback | Visual and location independent references |
| No reference ambiguity on the display | Multiple ways to refer to objects |

### 18.3.1.4. Technology Considerations

Speech recognition applications can be divided into many classes, as illustrated in Figure 18.7. The easiest category is the system that can only deal with isolated speech with limited vocabulary for a command-and-control application, where typically a context-free grammar is used to constrain the search space, resulting in a limited number of ways users can issue a command. The hardest is the one that can support channel- and speaker-independent continuous speech recognition with both dictation and robust application-specific understanding capability. This is also the one that can really add value to a wide range of speech applications. In Figure 18.7, a *command-and-control* application implies application-specific restrictive understanding capability—typically with less than 1000-word vocabulary and with a lower perplexity context-free grammar. A *dictation* application implies general-purpose dictation capability—typically with more than 60,000 words in the lexicon and a stochastic *n*-gram language model. A *dictation and SLU* application implies both general-purpose dictation and less restrictive application-specific spoken language understanding capability—typically with 60,000 words in the lexicon with both *n*-gram and semantic grammar language models for both recognition and understanding.

Although a number of parameters can be used to characterize the capability of speech recognition systems, user-centered design often dictates that you need to use the most challenging technology solutions. The speaking style (from isolated speech to continuous speech) can dramatically change the performance of a speech recognition system. Natural, spontaneous, continuous speech remains the most difficult to recognize; but it is more user friendly. Small vocabulary (less than 50 words) is generally more accurate than large vocabulary (greater than 20,000 words); but the small vocabulary limits the options the user has on the task. The lower-perplexity language model generally has a great recognition accuracy; but it imposes a considerable constraint on how the user can issue a command. Most systems may work in the high-SNR (>30 dB) environment; but it is desirable that the system should still work when the SNR is below 10 dB.

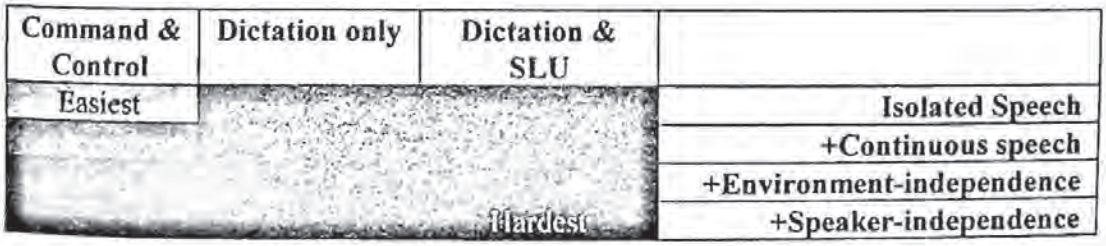



**Figure 18.7** Illustrative degrees of difficulty for different classes of speech recognition (gray level from easy to hard).

Associated with the input modality is the speech output modality for many speech applications. Speech as an output medium taxes user memory and may even cause loss of context and distraction from the task if it is too verbose. Most of the text-to-speech systems have a problem generating a prosody pattern that is dynamic and accurate enough to convey the underlying semantic message. Recorded speech still has the best quality; but it is generally inflexible in comparison to TTS. There are many ways to provide additional information to the TTS so that the final synthesized quality can be improved dramatically. Examples include the use of recorded speech for constant messages, transplanted prosody, and concept-to-speech synthesis as discussed in Chapters 16 and 17.

Improved quality and ease of use often need expensive authoring. This is true for speech recognition, synthesis, and understanding. For example, to support channel independence, you need not only additional channel normalization algorithms to deal with both additive and convolutional noises but also realistic training data to estimate model parameters. To support robust understanding, you need to author semantic grammars that have broad coverage. This often requires tedious authoring and labor-intensive data collection and annotation to augment the grammar. Ultimately, you need a modular and reusable system that can warrant the development costs. To make effective use of dialog and error-repair strategies, you need to adopt tools for preventing dialog design problems during the early stages, ones that guide the choice of words in dialog and feedback and ensure usability and correctness in the context, etc. In addition, as discussed in Chapter 17, whether you have a system-initiative dialog, user-initiative dialog, or mixed-initiative dialog will have significant ramifications on the authoring cost.

A spoken language application requires certain hardware and software on the user's computer in order to run.[13] Most new sound cards work well for speech recognition and text-to-speech, including Sound Blaster™. Some old sound cards are only *half duplex* (as opposed to *full duplex*). If a sound card is half duplex, it cannot record and play audio at the same time. Thus it cannot be used for barge-in and echo cancellation. The user can choose among different kinds of microphones: a close-talk headset microphone that is held close to the mouth, a medium-distance microphone, or a microphone array device that rests on the computer 30 to 60 centimeters away from the speaker. A headset microphone is often needed for noisy environments, although microphone array or blind separation techniques have the potential to close the gap in the future (see Chapter 10). Most speech products also need to calibrate the microphone gain by speaking one or two utterances, as illustrated in Figure 18.8. It adjusts the gain of the amplifier to make sure there is an appropriate gain without clipping the signal. This can be done in the background without distracting the user.

[13] Speech recognition engines currently on the market typically require a Pentium 200 or faster processor. Speech recognition for command and control consumes 1 to 4 MB RAM, and dictation requires an additional 16-32 MB. Text-to-speech engines use about 1-3 MB, but high-quality TTS can require more than 64 MB.

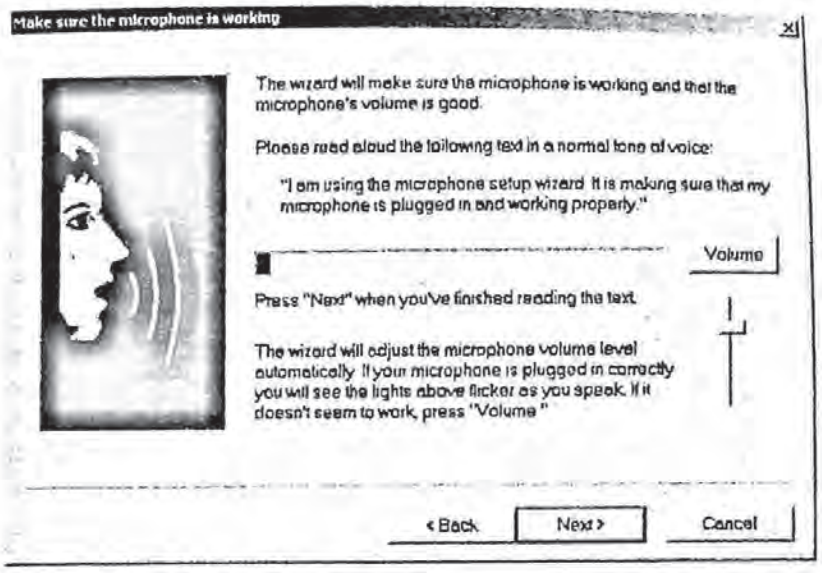



**Figure 18.8** Microphone setup wizard used in Microsoft Speech SDK 4.0. Reprinted with permission from Microsoft Corporation.

## 18.3.2. Handling Errors

Speech recognition and understanding can never be perfect. Current spoken language processing work is very much a matter of minimizing errors. Together with the underlying speech-engine technologies, the role of the interface is to help reduce the errors or minimize their severity and consequences. As discussed in earlier chapters, there are errors associated with word deletion, word insertion, and word substitution. The recognizer may be unable to reject noises and classify them as regular words. The speaker may also have new words that are not in the lexicon, or may have less clear articulation due to accent, stress, sickness, and excitement.

To repair errors in the system, we need to take each individual application into consideration, since the strategy is often application-dependent. To have a compelling application, we must design the application with both the limitation of the engine and the interaction of the engine and UI in mind. For example, multimodal applications are generally more compelling and useful than speech-only solutions as far as error handling is concerned. A multimodal application can overcome many limitations of today's SLU technologies, but it may be inappropriate for hands-busy or eyes-busy scenarios.

### 18.3.2.1. Error Detection and Correction

We need to detect errors before we can repair them. This is a particularly difficult problem that often requires high-level context and discourse information to decipher whether there are errors. We can certainly use confidence measures, as discussed in Chapter 9, to detect potential errors, although confidence measures alone are not particularly reliable.

There are many ways to correct the errors. The choice of which one to use depends on the types of the applications you are developing. We want to emphasize that correction based on another modality is often desirable, since it is unusual that two independent modalities make the same mistake. For example, speech recognition errors are often different from handwriting recognition errors. In addition to the complementary nature of these different errors, we can also leverage other modalities for improved error-correction efficacy. For example, a pen is particularly suitable for pointing, and we can use that to locate misrecognized words. A camera can be used to track whether the speaker is speaking in order to eliminate background noise generated from other sound sources.

Let's consider a concrete example. If you have a handheld device, you may be able to see the errors and respeak the desired commands or use a different modality such as pen to help task completion. When you want to create an email using speech, and a dictation error occurs, you can click the misrecognized word or phrases to display a list of $n$-best alternatives. You can select the correct word or phrase from the $n$-best list with the pen. Alternatively, you can respeak the highlighted word or phrase. Since the correct answer should not be the previous misrecognized word or phrase, you can exclude them in the second trial. If it is an isolated word in the correction stage, you can also use the correct silence context triphone models for the word boundary as the correct acoustic model. You can also use the correct $n$-gram language model from the left context with the assumption that the left words are all correct. With these correct acoustic and language models, it is expected that the second trial would have a better chance to get the right answer.

If you have telephony-based applications, you can count on keywords such as *no*, *I meant* to identify possible recognition or understanding errors. When errors occur, you should provide specific error-handling messages so that the user knows what response the system wants. An adequate error message should tell the user what is wrong, how to correct it, and where to get help if needed. Let's consider an example of the following directory assistant application:

**Computer: Please speak the name**

**User: Mike**

**Computer: Please speak the name**

**User: Mike**

Clearly, the user does not know what responses the computer wants. An improved dialog can be:

**Computer: Please speak the first and last name**

**User: Mike Miller**

**Computer: Which division is Mike Miller working?**

**User: Research group**

You can use alternative guesses intelligently for telephony applications, too. For example, if the user says "*Mike Miller*" and the computer is unsure of what was said, it can prompt the user to repeat with a constrained grammar. In the example above, if the computer is unsure whether the user said Mike or Mark, it may want to respond with "*Did you say Mike Miller or Mark Miller*?" before it proceeds to clarify which division Miller is working in.

When an error is detected, the system should be adapted to learn from the error. The best error-repair strategy is to use an adaptive acoustic and language model so that the system can learn from the errors and not repeat the same mistake again. This is why adaptive acoustic and language modeling, as discussed in Chapters 9 and 11, is critical for real acceptance of speech applications. When an error occurs and the computer understands the error via user correction, you can use the corrected words or phrases to adapt related components in the system.

### 18.3.2.2. Feedback and Confirmation

Users need feedback from the system so that they understand the status of the interaction. When a command is issued, the user expects the system to acknowledge that the command is heard. It is also important to tell the users if the system is busy so that they know when to wait. Otherwise, the users may interpret the lack of response or feedback as errors of recognition or understanding.

If the user has requested that an action be taken, the best response is to carry out the action as requested, but if the action cannot be carried out, the user must be notified. Sometimes, the user may need this feedback to know whether it is her turn to speak again. If the interface is multimodal, you can use visual feedback to indicate the status of the interaction. For example, in Microsoft's MiPad research prototype, when the microphone is activated with a pen pointing to a specific field, the microphone levels are indicated in the feedback rectangle in strip-chart fashion as shown in Figure 18.9 for the corresponding field. Initially this wipes from left to right, but when the right edge is reached, the contents scroll to the left as each new sample is added. The height of each sample indicates the audio level; there is some indication given for the redline level for a possible saturation. It also shows the total number of frames as well as the number of remaining frames to be processed. Then, as the



**Figure 18.9** MiPad's dynamic volume meter. Reprinted with permission from Microsoft Corporation.

recognizer processes, the black bar wipes from left to right to indicate its progress. When the recognizer completes (or is canceled by the user), the dynamic volume meter is removed. The volume meter feedback described addresses a number of issues that are critical to most users: if the computer is listening or not; if the volume is too high or not, and when the computer finishes processing the speech data.

If the interface has no visual component, you can use auditory cues or speech feedback. If there is more than a 3-second delay after the user issues a command and the system responds, an auditory icon during the delay, such as music, may be used. A delay of less than 2-3 seconds is acceptable without using any feedback.

Since there are errors in spoken language systems, you may have to use confirmation questions to assure that the computer heard the correct message. You can associate risks with different commands, so you can have a different level of strategy for confirmation. For example, *format disk* should have a higher threshold or may not be allowed by voice at all. You can have an explicit confirmation before executing the command. You can also have implicit confirmation based on the level of the risks in the corresponding confidence scores from the recognizer. For example, if the user's response has more than one possible meaning (X and Y) or the computer is not confident about the recognized result (X or Y), the computer could ask, "*Do you want to do X or Y?*" You want to be specific about what the system needs. A prompt like "*Do you mean X or Y?*" is always better than *Please repeat.* You have to balance the cost of making an error with the extra time and annoyance in requiring the user to confirm a number of statements.

In a telephony-based application, you need to tell the users specifically that they are talking to a computer, not a real person. When AT&T used a long greeting for its telephone customer service, their prompts were as follows:

*"AT&T Automated Customer Service. This service listens to your speech and sends your call to the appropriate operator. How may I help you?"*

The longer version resulted in shorter utterances from the people calling in. The short automated version did not help shorten the utterances, because people did not seem to catch the automated connotation.

You can also design the prompt in such a way that you can constrain how people respond or use their words. Let's consider an example of the following directory assistant application:

**Computer: Welcome to Directory Assistant. We look forward to serving you. How can I help you?**

**User: I am interested in knowing if you can tell me the office number of Mr. Derek Smith?**

To have the user follow the lead of the computer, an improved dialog can be:

**Computer: Welcome to Directory Assistant. Please say the full name of the person.**

**User: Derek Smith**

When the system expects the user to respond with both the first name and last name, you should still design the grammar so that the system can recognize a partial name with lower probabilities. This makes the system far more robust—a good example of combining engine technology and interface seamlessly for improved user satisfaction.

## 18.3.3. Other Considerations

When speech is used as a modality, you should remember the following general principles:

### 18.3.3.1. Don't Use Speech as an Add-On Feature

It's poor design to just bolt speech recognition onto an application that is designed for a mouse and keyboard. Such applications get little benefit from speech recognition. Speech recognition is not a replacement for the keyboard and mouse, but in some circumstances it is a better input device than both. Speech recognition makes a terrible pointing device, just as the mouse makes a terrible text entry device, or the keyboard is bad for drawing. When speech recognition systems were first bolted onto the PC, it was thought that speaking menu names would be really useful. As it turns out, very few users use speech recognition to access a window menu.

### 18.3.3.2. Give Users Options

Every feature in an application should be accessible from all input devices: keyboard, mouse, and speech recognition. Users naturally use whichever input mechanism provides them the quickest or easiest access to the feature. The ideal input device for a given feature may vary from user to user.

### 18.3.3.3. Respect Technology Limitations

There are a number of examples of poor use of speech recognition. Having a user spell out words is a bad idea for most recognizers, because they are too inaccurate unless you constrain the recognizer for spelling. An engine generally has a hard time detecting multiple speakers talking over each other in the same digital-audio stream. This means that a dictation system used to transcribe a meeting will not perform accurately during times when two or more people are talking at once. An engine cannot hear a new word and guess its spelling unless the word is specified in the vocabulary. Speakers with accents or those speaking in

nonstandard dialects can expect more errors until they train the engine to recognize their speech. Even then, the engine accuracy will not be as high as it would be for someone with the expected accent or dialect. An engine can be designed to recognize different accents or dialects, but this requires almost as much effort as porting the engine to a new language.

#### 18.3.3.4. Manage User Expectations

When you design a speech recognition application, it is important to communicate to the user that your application is speech aware and to provide him or her with the commands it understands. It is also important to provide command sets that are consistent and complete. When users hear that they can speak to their computers, they instantly think of *Star Trek* and *2001: A Space Odyssey*, expecting that the computer will correctly transcribe every word that they speak, understand it, and then act upon it in an intelligent manner. You should also convey as clearly as possible exactly what an application can and cannot do and emphasize that the user should speak clearly, using words the application understands.

### 18.3.4. Dialog Flow

Dialog flow is as important to speech interface as screen design is to GUI. A robust dialog system is a prerequisite to the success of the speech interface. The dialog flow design and the prompting strategy can dramatically improve user experience and reduce the task complexity of the spoken language system.

#### 18.3.4.1. Spoken Menus

Many speech-only systems rely on tree-structure spoken menus as the main navigation vehicle when no sophisticated dialog system is used. The design of these menus is, therefore, critical to the usability of the system. If there are only two options, you can have "*Say 1 for yes, or 2 for no.*" However, if a menu has more than two options, you should describe the result before you specify the menu option. For example, you want to have the system say "*For email say 1. For address book say 2...*" instead of "*Say 1 for email. Say 2 for address book....*" This is because the user may forget which choice he wants before hearing all the menu options.

You should not use more than five options at each level. If there are more than five, you should consider submenus. It is important to place the most frequently used options at the top of the menu hierarchy.

You should also design the menu so that commands sound different. As a rule of thumb, the more different phonemes you have between two commands, the more different they sound to the computer. The two commands, *go* and *no*, only differ by one phoneme, so when the user says, "*Go*" the computer is likely to recognize "*No.*" However, if the commands were "*Go there*" and "*No way*" instead, recognition would be much easier.

### 18.3.4.2. Prompting Strategy

When you design the prompt, use as few words as possible. You should also avoid using a personal pronoun when asking the user to respond to a question. This gives the user less to remember, as speech output is slow and one-dimensional. Typical users are able to remember three to four menu or prompt options at a time. You should also allow users to interrupt the computer with *barge-in* techniques, as discussed in Chapter 10.

It is often a good idea to use small steps to query the user at each step if application has a speech-only interface progressively. You can start with short high-level prompts such as "*How can I help you?*" If the user does not respond appropriately, the system can provide a more detailed prompt such as "*You can check your email, calendar, address book, and your home page.*" If the user still does not respond correctly, you can give a more detailed response to tell the user how to speak the appropriate commands that the computer can understand.

If the application contains users' personal information, you may want to treat novice users and experienced users differently. Since the novice users are unfamiliar with the system and dialog flow, you can use more detailed instructions and then reduce the number of words used the next time when the user is going through the same scenario.

### 18.3.4.3. Prosody

For TTS, prosodic features, including pitch, volume, speed, and pause, are very important for the overall user experience. You can vary pitch and volume to introduce new topics and emphasize important sections. Increased pitch dynamic range also makes the system sound lively.

Users tend to mimic the speed of the system. If the system is reacting and speaking slowly, the user may tend to slow down. If the computer is speaking quickly, the user may speed up. An appropriate speech rate for natural English speech is about 150 to 170 words per minute.

Pauses are important in conversation, but they can be ambiguous, too. We generally use pauses to suggest that it is the user's turn to act. On the other hand, if the system has requested a response from the user and the user has not responded, you should use the time-out period (typically 10 seconds) to take appropriate actions, which may be a repeat of a more detailed prompt or may be a connection to an operator. People are generally uncomfortable with long pauses in conversation. If there are long pauses in the dialog flow, people tend to talk more and use less meaningful words, resulting in more potential errors.

## 18.4. INTERNATIONALIZATION

To meet the demands of international markets you need to support a number of languages. For example, Microsoft Windows 2000 supports more than 40 languages. Software interna-

tionalization typically involves both globalization and localization phases. Globalization is the process of defining and developing a product such that its core features and code design do not make assumptions about a single language. To adapt the application for a specific market, you need to modify the interface, resize the dialog boxes, and translate text to a specific market. This process is called localization. The globalization phase of software internationalization typically focuses on the design and development of the product. The product designers plan for locale-neutral features to support multicultural conventions such as address formats and monetary units. Development of the source code is typically based on a single code base to have the capability to turn locale-specific features on and off without modifying the source code, and the ability to correctly process different character encoding methods. Products developed with the single worldwide binary model typically include complete global functionality, such as support for Input Method Editors used in Asian languages and bidirectional support for Arabic and Hebrew languages.

In addition to the most noticeable change in the translated UI, spoken language engines such as speech recognition or text-to-speech synthesis require significant undertakings. You need to have not only the lexicon defined for the specific language but also a large amount of speech and text data to build your acoustic and language model. There are many exceptional rules for different languages. Let's take dictation application as an example. When you have inverse text normalization, whole numbers may be grouped differently from country to country, as shown in Table 18.3. In the United States, we group three digits separated by a comma. However, many European locales use a period as the number separator, and Chinese conventions typically do not group numbers at all.

**Table 18.3** Examples of how numbers are grouped by different locales for inverse text normalization.

| | |
|---|---|
| 123,456,789.0 | United States and most locales |
| 123456789.0 | Chinese |
| 12,34,56,789,0 | Hindi |

For the speech recognition engine localization, there are a number of excellent studies on both Asian [5, 15, 16, 21, 24, 32, 34, 37] and European languages [7, 8, 20, 22, 27, 31, 40]. Many spoken language processing components, such as the speech signal processing module, the speech decoder, and the parser, are language independent. Major changes are in the *content* of the engine, such as the lexicon and its associated processing algorithm, the grammar, and the acoustic model or language model parameter.

In general, it is easier to localize speech recognition than either text-to-speech or spoken language understanding. This is because a speech recognition engine can be mostly automatically derived from a large amount of speech and text data, as discussed in Chapters 9 through 13. For Chinese, you need to have special processing for tones, and the signal-processing component needs to be modified accordingly [4]. For both Chinese and Japanese the lexicon also needs to be carefully selected, since there is no space between words in the lexicon [12, 46]. As discussed in Chapter 17, the parser can be used without much modifica-

tion for SLU systems. Most of localization efforts would be on the semantic/syntactic grammar and dialog design, which is language specific by nature.

As discussed in Chapters 14 and 15, the TTS front end and symbolic prosody components are much harder to localize than the TTS back end. The TTS back end can be easily localized if you use the trainable approach discussed in Chapter 16 on the assumption that the speech recognizer for the language is available. The internationalization process mainly consists of creating tools that can generate training phrase lists from target language texts and dictionaries, and ensuring that any reasonable-sized set of symbolic distinctive language sound inventories (phone sets) can be accepted by the system. The human vocal tract is largely invariant across language communities, and this relative invariance is reflected in the universality of the speech synthesis tools and methods.

When the lexical, syntactic, and semantic content is localized, you should be aware that the incorrect use of sensitive terms can lead to outright product bans in some countries, while other misuses are offensive to the local culture. For example, Turkey forbids the use of the term *Kurdistan* in text and any association of Kurdistan with Turkey. Knowing proper terminology is critical to the success of your application internationalization.

## 18.5. CASE STUDY—MIPAD

An effective speech interface design requires a rigorous process that typically consists of three steps with a number of iterations based on the user evaluation:

- Specifying the application
- Rapid prototyping
- Evaluation

We use Microsoft's research prototype MiPad [18] as an example to illustrate how you can use the process to create an effective speech application. As a wireless Personal Digital Assistant (PDA), MiPad fully integrates continuous speech recognition (CSR) and spoken language understanding (SLU) to enable users to accomplish many common tasks using a multimodal interface and wireless technologies. It tries to solve the problem of pecking with tiny styluses or typing on minuscule keyboards in today's PDAs or smart phones. It also avoids the problem of being a cellular telephone that depends on speech-only interaction. MiPad incorporates a built-in microphone that activates whenever a field is selected. As a user taps the screen or uses a built-in roller to navigate, the tapping action narrows the number of possible instructions for spoken language processing. MiPad runs on a Windows CE Pocket PC with a Windows 2000 Server where speech recognition is performed. The CSR engine has a 64,000 word vocabulary with a unified context-free grammar and $n$-gram language model as discussed in Chapter 11. The SLU engine is based on a robust chart parser and a plan-based dialog manager discussed in Chapter 17.

### 18.5.1. Specifying the Application

The first step to develop an application is to identify key features that can help users. You need to understand what work has been done, and how that work can be used or modified. You need to ask yourself why you need to use speech for the applications and review project goals and time frames. The first deliverable will be the spec of the application that includes a documented interface design and usability engineering project plan. Some of the key components should include:

- Define usability goal, not technology
- Identify the user groups and characteristics, interview potential users, and conduct focus-group studies
- Create system technology feasibility reports
- Specify system requirements and application architecture
- Develop development plan, QA test plan, and marketing plan

You should develop task scenarios that adopt a user's point of view. Scenarios should provide detail for each user task and be in a prose or dialog format. You may also want to include frequency information as a percentage to illustrate whether a particular case is happening more than 20% or 80% of the time, which helps you focus on the most common features first.

To create a conceptual interaction model, not the software architecture, we need to have a mental model for the users. Users have to manipulate interface objects either literally or figuratively in the navigation. Interfaces without clear and consistent user objects are difficult to use. Generally, you need a script flow diagram if speech is used. This describes how the user moves through dialogs with the system. To create a script flow diagram from your scenarios, you can start at the beginning of a scenario and derive a set of possible dialogs the user needs to have with the system to complete the scenario. You can draw a box representing a dialog with arrows that shows the order and relationship among the dialogs. If your interface involves graphic display, you need to include window flow diagrams as well, which organize objects and views and define how the user moves from one object to another. Of course, these two diagrams may be combined for effective multimodal interaction.

When the initial spec is in place, you can use the conceptual model to create actual paper windows and dialogs for the interface. You must iterate the design several times and get feedback early. Paper prototyping allows you to do these tasks most efficiently. It is useful to conduct walkthroughs with a number of people as though they were performing the task. You may want to start usability testing to collect real data on real users before the software is implemented. Even a system that is well designed and follows the best proactive approach may reveal some usability problems during testing. Usability testing has established protocols, procedures, and methodologies to get reliable data for design feedback [36, 42]. Generally you need to decide what to test, create usability specifications for the tasks, select the

right user groups, specify exact testing scenarios, conduct the actual tests with participants, analyze the data, and report critical findings.

How did we do this for MiPad? Since MiPad is a small handheld device, the present pen-based methods for getting text into a PDA (Graffiti, Jot, soft keyboard) are barriers to broad market acceptance. As an input modality, speech is generally not as precise as mouse or pen to perform position-related operations. Speech interaction can be adversely affected by the ambient noise. When privacy is of concern, speech is also disadvantageous, since others can overhear the conversation. Despite these disadvantages, speech communication is not only natural but also provides a powerful complementary modality to enhance the pen-based interface. Because of these unique features, we need to leverage the strengths and overcome the technology limitations that are associated with the speech modality. Pen and speech can be complementary and they can be used very effectively for handheld devices. You can tap to activate a microphone and select appropriate context for speech recognition. The advantage of pen is typically the weakness of speech and vice versa. This implies that user interface performance and acceptance could increase by combining both. Thus, visible, limited, and simple actions can be enhanced by nonvisible, unlimited, and complex actions.

**Table 18.4** Benefits of having speech and pen for MiPad.

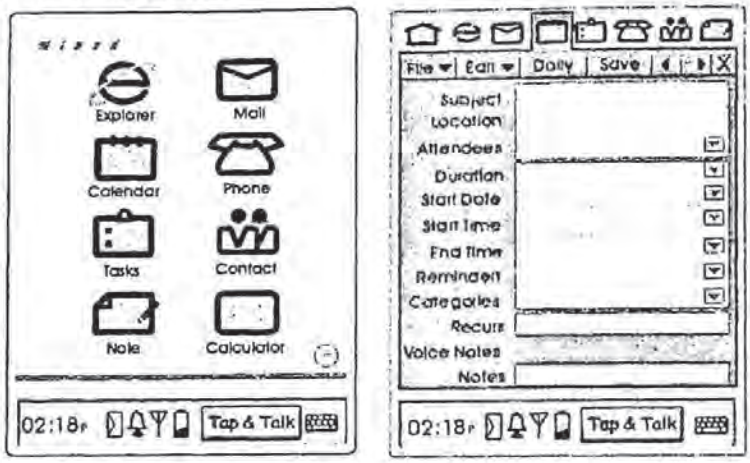
| Action | Benefit |
|---|---|
| Ed uses MiPad to read an email, which reminds him to schedule a meeting. Ed taps to activate microphone and says *Meet with Peter on Friday.* | Using speech, information can be accessed directly, even if not visible. Tap and Talk also provides increased reliability for speech detection. |
| The screen shows a new appointment to meet with Peter at 10:00 on Friday for an hour. | An action and multiple parameters can be specified in only a few words. |
| Ed taps Time field and says *Noon to one thirty.* | Field values can be easily changed using field-specific language and semantic models. |
| Ed taps Subject field dictates and corrects a couple of sentences explaining the purpose of the meeting. | Bulk text can be entered easily and faster. |

People tend to like to use speech to enter data and pen for corrections and pointing. As illustrated in Table 18.4, MiPad's Tap and Talk interface offers a number of benefits. MiPad has a Tap and Talk field that is always present on the screen, as illustrated in MiPad's start page in Figure 18.10 (a) (the bottom gray window is always on the screen).

The user can give spontaneous commands by tapping the Tap and Talk field and talking to it. The system recognizes and parses the command, such as showing a new appointment form. The appointment form shown on MiPad's display is similar to the underlying semantic objects. The user can have conversation by *tapping and talking* to any subfield as well. By tapping to the attendees field in the calendar card shown in Figure 18.10 (b), for example, the semantic information related to potential attendees is used to constrain both CSR and SLU, leading to a significantly reduced error rate and dramatically improved throughput. This is because the perplexity is much smaller for each subfield-dependent lan-

guage and semantic model. General text fields, such as the title or body of an email, call for general dictation. Dictation is handled by the same mechanism as other speech entries: The user dictates while tapping the pen where the words should go. To correct a misrecognized word, a tap on the word invokes a correction window, which contains the *n*-best list and a soft keyboard. The user may hold the pen on the misrecognized word to dictate its replacement.

From the start page, you can reach most application states. There are also a number of short-cut states that can be reached via the hardware buttons. The design is refined incrementally in a number of iterations.




(a) (b)

**Figure 18.10** Concept design for (a) MiPad's main card and (b) MiPad's calendar card.

## 18.5.2. Rapid Prototyping

When you have the spec for the application, you need to communicate the design to the development team so they can rapidly develop a prototype for further iterations. In practice, handing off an interface to a development team without involvement in its implementation seldom works. This is because the developers always have questions about the design document. You need to be available to them to interpret your design. Technical and practical issues may also come up during development, requiring changes to the interface design.

For speech applications, there are a number of key components that have a significant implication for the user interface design. As your design iterations progress, it is useful to track the changes in these components and the effect that these changes have on the user experience of the prototype.

- **Semantic classes**: Each semantic class, as discussed in Chapter 17, is representing real-world entities or associated with the action that applications can take. We need to abstract the application to a level such that we can represent all the actions that the application can take with a number of semantic classes.

Each semantic class has needed slots that require a context-free grammar with a decent coverage. You can start with example sentences and reuse some of the predefined task-independent CFGs, such as date and time. The interface design can help to constrain users to say the sentences that are within the domain.

- **Dialog control**: When appropriate semantic objects are selected, you need to decide the flow of these semantic objects with an appropriate dialog strategy, either speech-based or GUI-based, which should include both inter- and intra-frame control and error-repair strategy. In addition, appropriate use of confidence measures is also critical.
- **Language models**: You need to use appropriate language models for the speech recognizer. Context-free grammars used for the semantic classes can be used for the recognizer. As coverage is generally limited, it is desirable to use the *n*-gram as an alternative. You can use a unified context-free grammar and *n*-gram for improved robustness, as introduced in Chapter 11.
- **Acoustic models**: In comparison to the semantic classes and language models, subphonetic acoustic models can be used relatively effectively for task-independent prototyping. Nevertheless, mismatches such as microphones, environment, vocabulary, and speakers are critical to the overall performance. You may want to consider speaker-dependent or environment-dependent prototyping initially, as this minimizes the impact of various mismatches.
- **TTS models**: If the speech output is well defined, you can adapt the general-purpose TTS systems with the correct prosody patterns. This can be achieved with the transplanted prosody in which variables can be reserved for TTS to deal only with proper nouns or phrases that cannot be predecided. You can also adapt acoustic units such that the concatenation errors can be minimized.

The work needed for rapid prototyping is significant, but less than it might at first appear to be. Earlier tasks can be repeated in later iterations when more data become available in order to further improve the usability of the overall system.

### 18.5.3. Evaluation

It is vital to have a rigorous evaluation and testing program to measure the usability of the prototype. It is a good practice to evaluate not only the entire system but also each individual component. The most important measure should be whether users are satisfied with the entire system. You can ask questions such as: *Is the task completion time much better? Is it easier to get the job done? Did we solve the major problem existing in the previous design*? Most of the time, we may find that the overall system can't meet your goal. This is why you need to further analyze your design and evaluate each individual component from UI to the underlying engines.

Before the prototype becomes fully functioning, you may want to use Wizard-of-Oz (WOZ) experimentation to study the impact of the prototype. That is, you have a real person (the wizard) to control some of the functions that you have not implemented yet. The person can carry out spoken interactions with users, who are made to believe that they are interacting with a real computer system. The goal of WOZ is to study the behavior of human-computer interactions. By producing data on the interaction between a simulated system and its users, WOZ provides the basis for early tests of the system and its feasibility prior to implementation. It can also simulate the error patterns in the existing spoken language system.

WOZ is still expensive, as you have the cost of training a wizard in addition to the usual costs of selecting experimental subjects, transcribing the session, and analyzing overall results. The use of WOZ has so far been justified in terms of relatively higher cost of having to revise an already implemented system that turned out to be flawed in the usability study.

In practice, WOZ may be replaced by the implementation-test-revise approach based on available tools. Whether or not WOZ is preferable depends on several factors, such as the methods and tools available, the complexity of the application to be designed, and the risk and cost of implementation failure. Low complexity speaks in favor of directly implementing the interface and application without interposing a simulation phase. On the other hand, high complexity may advocate iterative simulations before a full-fledged implementation.

To evaluate each component, you often need to select users that typify a wide range of potential users. You can collect representative data from these typical users based on the prototype we have. The most important measures include language-model perplexity, speech recognition accuracy and speed, parser accuracy and speed (on text and speech recognition output), dialog control performance, prompting strategy and quality, error-repair efficacy, impact of adaptive modeling, and interactions among all these components.

Effective evaluation of a user interface depends on a mix of objective and subjective measures. One of the most important characteristics is repeatability of your results. If you can repeat them, then any difference is likely due to the changes in the design that you are testing. You will want to establish reference tasks for your application and then measure the time that it takes subjects to complete each of those tasks. The consistent selection of subjects is important; as is the use of enough subjects to account for individual differences in the way they complete the tasks.

How do we evaluate MiPad? For our preliminary user study, we did a benchmark study to assess the performance in terms of task-completion time, text throughput (WPM), and user satisfaction. The focal question of this study is whether the MiPad interface can provide added value to the existing PDA interface.

*Is the task-completion time much better?* We had 20 computer-savvy users test the partially implemented MiPad prototype. These people had no experience with PDA or speech recognition software. The tasks we evaluated include creating a new e-mail, checking a calendar, and creating a new appointment. Task order was randomized. We alternated tasks to different user groups using either pen-only or Tap and Talk interfaces. The text

throughput was calculated during email paragraph transcription tasks.[14] Compared to using the pen-only iPaq interface, we observed that the Tap and Talk interface is about twice as fast transcribing email.[15] For the overall command-and-control operations such as scheduling appointments, the Tap and Talk interface was about 33% faster than the existing pen-only interface.[16] Error correction for the Tap and Talk interface was one of the most unsatisfied features. In our user study, calendar access time using the Tap and Talk methods was about the same as pen-only methods, which suggests that simple actions are very suitable for pen-based interaction.

*Is it easier to get the job done?* Most users we tested stated that they preferred using the Tap and Talk interface. The preferences were consistent with the task completion times. Indeed, most users' comments concerning preference were based on ease of use and time to complete the task, as shown in Figure 18.11.

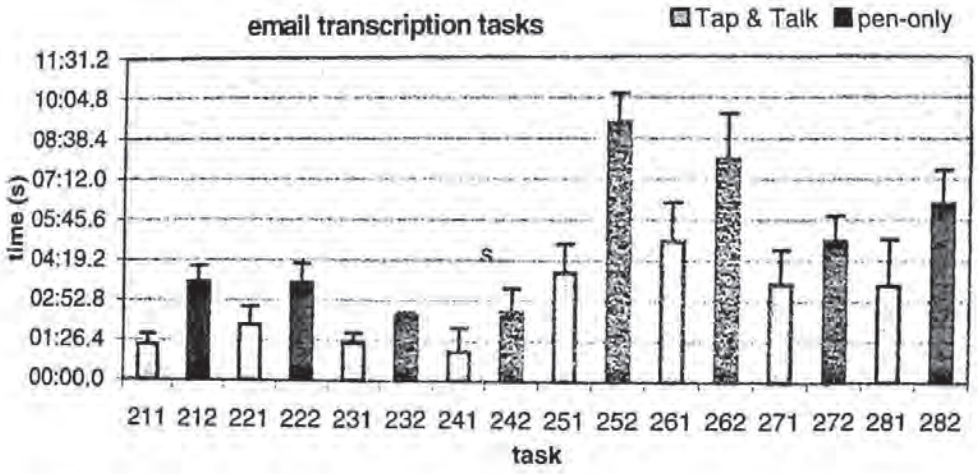



**Figure 18.11** Task completion time of email transcription between the pen-only interface and Tap and Talk interface.

## 18.5.4. Iterations

With rigorous evaluations, you can revisit the components that are on the critical path. In practice, you need to run several iterations to improve user interface design, semantic classes, CFGs, *n*-gram, dialog control, and acoustic models. Iterative experiments are particularly important for deployment, as the coverage of the grammars can be dramatically improved only after we accumulate enough realistic data from end users.

[14] Transcription may not be a realistic task and gives an advantage to speech, because speech avoids a lot of attention shifting that is involved with pen input.

[15] The corresponding speech recognition error rate for the tasks is about 14%, which is based on using a close-talk microphone and a speaker-dependent acoustic model trained from about 20 minutes of speech.

[16] The SLU (at card level) error rate for the task is about 4%.

For MiPad, the performance improvement from using real data has been more dramatic than any other new technologies we can think of. When we collected domain-specific data, MiPad's SLU error rate was reduced by more than a factor of two. The major improvement came from broadening the semantic grammars from real data. We believe that data collection and iterative improvement should be an integral part of the process of developing interactive systems.

Despite unquestionable progress in the last decade, there are many unsolved problems in the procedures, concepts, theory, methods, and tools. Designing an effective speech application and interface remains as much of an art as it is an exact science with established procedures of good engineering practice. The quest for an effective speech application has been the goal for many speech and interface researchers. A successful speech application needs to significantly improve people's productivity. We believe that speech will be one of the most important technology components to enable people access information more effectively in the near future.

MiPad is a work in progress to develop a consistent interaction model and engine technologies for three broad classes of applications. A number of discussed features are yet to be fully implemented and iteratively tested. Our currently tested features include PIM functions only. Despite our incomplete implementation, we observed in our preliminary user study that speech and pen have the potential to significantly improve user experience. Thanks to the multimodal interaction, MiPad also offers a far more compelling user experience than standard telephony interaction. The success of MiPad depends on spoken language technology and always-on wireless connection. With upcoming 3G wireless deployments in sight,[17] the critical challenge for MiPad remains the accuracy and efficiency of our spoken language systems, since MiPad may be used in noisy environments without a close-talk microphone, and the server also needs to support a large number of MiPad clients.

## 18.6. HISTORICAL PERSPECTIVE AND FURTHER READING

User interface design is particularly resistant to rigid methodologies that attempt to define it simply as another branch of engineering. This is because design deals with human perception and human performance. Issues affecting the user interface design typically include a wide range of topics such as technology capability and limitations, product productivity (speed, ease of use, range of functions, flexibility), esthetics (look and feel, user familiarity, user impression), ergonomics (cognitive load, memory), and user education [2, 13, 14, 30, 35, 38, 44, 45]. The spoken language interface is a unique and specialized human interface medium that differs considerably from the traditional graphical user interface, and there are a number of excellent books discuss how to build an effective speech interface [1, 25, 39, 43].

When we communicate with speech, we must hear and comprehend all information sequentially. This very serial or sequential nature makes speech less effective as an information presentation medium than the graphics. As an input medium, speech must be supplied

[17] http://www.wirelessweek.com/issues/3G

one command or word at a time, requiring the user to remember and order information in a way that is more taxing than it is with visual and mechanical interfaces. The powerful and parallel presentation of information in today's GUI in the form of menus, icons, and windows makes GUI one of the most effective media for people to communicate. GUI is not only parallel but also persistent. That is, once presented, information remains on the screen until replaced as desired. Speech has no such luxury, which places the burden of remembering machine output onto the user. It is no wonder that *a picture is worth a thousand words*. We believe this is one of the key reasons why the spoken language interface has not taken off in comparison to the graphical user interface.

Although speech is natural, as humans already know how to talk effortlessly, this common shared experience of humans may not necessarily translate to human-machine interaction. This is because machines do not share a common cultural heritage with humans and they do not posses certain assumptions about the reality of the world in which we live. We have no precedent for effective speech interaction with nonsentient devices that are self-aware. Our experience with natural speech interaction is based on the assumption that when we talk to another person, the other person has some stake in the outcome of the interaction. The result of this expectation is that structured and goal-oriented conventions are necessary to steer people away from social speech behaviors toward task-oriented protocols, which essentially eliminate our expectation of naturalness.

The fact that speech interactions are one dimensional and time consuming has significant ramifications for interface design. The major challenge is to help the user accomplish as much as possible in as little time as possible. The designer thus must encourage the user to construct a mental model of the task and the interface that serves it, thereby creating an illusion of forward movement that exploits time by managing and responding to the user goals. Many telephony applications have carefully considered these constraints and provided a viable alternative to GUI-based applications and interfaces.

Despite the challenges, a number of excellent applications set milestones for the spoken language industry. Widely used examples include DECTalk (TTS), AT&T universal card services, Dragon's NaturallySpeaking dictation software, and Microsoft Windows 2000 TTS accessibility services. With the wireless smart devices such as smart phones, we are confident that MiPad-like devices will find their way to empower people to access information anywhere and anytime, leading to dramatically improved productivity.

There are a number of companies offering a variety of new speech products. Their Web sites are the best reference points for further reading. The following Web sites contain relevant spoken language applications and user interface technologies:

- AOL (http://www.aol.com)
- Apple (http://www.apple.com/macos/speech)
- AT&T (http://www.att.com/aspg)
- BBN (http://www.bbn.com)
- Dragon (http://www.dragonsystems.com)
- IBM (http://www.ibm.com/software/speech)

- Lernout & Hauspie (http://www.lhs.com)
- Lucent Bell Labs (http://www.lucent.com/speech)
- Microsoft (http://www.microsoft.com/speech)
- Nuance (http://www.nuance.com)
- NTT (http://www.ntt.com)
- Philips (http://www.speech.be.philips.com)
- Speechworks (http://www.speechworks.com)
- Tellme (http://www.tellme.com)
- Wildfire (http://www.wildfire.com)

## REFERENCES

[1] Balentine, B. and D. Morgan, *How to Build a Speech Recognition Application*, 1999, Enterprise Integration Group.

[2] Beyer, H. and K. Holtzblatt, *Contextual Design: Defining Customer-Centered Systems*, 1998, San Francisco, Morgan Kaufmann Publishers.

[3] Campbell, J., "Speaker Recognition: A Tutorial," *Proc. of the IEEE*, 1997, **85**(9), pp. 1437-1462.

[4] Chang, E., *et al.*, "Large Vocabulary Mandarin Speech Recognition with Different Approaches in Modeling Tones," *Int. Conf. on Spoken Language Processing*, 2000, Beijing, China.

[5] Chien, L.F., K.J. Chen, and L.S. Lee, "A Best-First Language Processing Model Integrating the Unification Grammar and Markov Language Model for Speech Recognition Applications," *IEEE Trans. on Speech and Audio Processing*, 1993, **1**(2), pp. 221-240.

[6] Dobler, S., "Speech Control In The Mobile Communications Environment," *IEEE ASRU Workshop*, 1999, Keystone, CO.

[7] Dugast, C., X. Aubert, and R. Kneser, "The Philips Large-Vocabulary Recognition System for American English, French and German," *Proc. of the European Conf. on Speech Communication and Technology*, 1995, Madrid, pp. 197-200.

[8] Eichner, M. and M. Wolff, "Data-Driven Generation of Pronunciation Dictionaries In The German Verbmobil Project - Discussion of Experimental Results," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, pp. 1687-1690.

[9] Furui, S., "Recent Advances in Speaker Recognition," *Pattern Recognition Letters*, 1997, **18**, pp. 859-872.

[10] Galitz, W.O., *Handbook of Screen Format Design*, 1985, Wellesley, MA, Q. E. D. Information Sciences Inc.

[11] Galitz, W.O., *User-Interface Screen Design*, 1993, Wellesley, MA, Q. E. D. Information Sciences Inc.

[12] Gao, J., *et al.*, "A Unified Approach to Statistical Language Modeling for Chinese," *Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, pp. 1703-1706.

[13] Hackos, J. and J. Redish, *User and Task Anaysis for User Interface Design*, 1998, New York, John Wiley.

[14] Helander, M., *Handbook of Human-Computer Interaction*, 1997, Amsterdam, North-Holland.

[15] Hon, H.W., *et al.*, "Towards Large Vocabulary Mandarin Chinese Speech Recognition," *Int. Conf. on Acoustics, Signal and Speech Processing*, 1994, Adelaide, Australia, pp. 545-548.

[16] Hon, H.-W., Y. Ju, and K. Otani, "Japanese Large-Vocabulary Continuous Speech Recognition System Based on Microsoft Whisper," *The 5th Int. Conf. on Spoken Language Processing*, 1998, Sydney, Australia.

[17] Horton, W., *The Icon Book*, 1994, New York, John Wiley.

[18] Huang, X., *et al.*, "MIPAD: A Next Generation PDA Prototype," *Int. Conf. on Spoken Language Processing*, 2000, Beijing, China.

[19] Karat, C., *et al.*, "Patterns of Entry and Correction in Large Vocabulary Continuous Speech Recognition Systems," *CHI'99*, 1999, Pittsburgh, pp. 15-20.

[20] Karat, J., "Int. Perspectives: Some Thoughts on Differences between North American and European HCI," *ACM SIGCHI Bulletin*, 1991, **23**(4), pp. 9-10.

[21] Kumamoto, T. and A. Ito, "Structural Analysis of Spoken Japanese Language and its Application to Communicative Intention Recognition," *Proc. of the Fifth Int. Conf. on Human-Computer Interaction—Poster Sessions: Abridged Proc.*, 1993, pp. 284.

[22] Lamel, L. and R.D. Mori, "Speech Recognition of European Languages" in *Proc. IEEE Automatic Speech Recognition Workshop* 1995, Snowbird, UT, pp. 51-54.

[23] Laurila, K. and P. Haavisto, "Name Dialing—How Useful Is It?," *IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 2000, Istanbul, pp. 3731-3734.

[24] Lyu, R.Y., *et al.*, "Golden Mandarin (III)—A User-Adaptive Prosodic Segment-Based Mandarin Dictation Machine for Chinese Language with Very Large Vocabulary," *Proc. of the IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, 1995, Detroit, pp. 57-60.

[25] Markowitz, J., *Using Speech Recognition*, 1996, Upper Saddle River, Prentice Hall.

[26] Martin, G.L., "The Utility of Speech Input in User-Computer Interfaces," *Int. Journal of Man-Machine Studies*, 1989, **30**(4), pp. 355-375.

[27] Möbius, B., "Analysis and Synthesis of German F0 Contours by Means of Fujisaki's Model," *Speech Communication*, 1993, **13**(53-61).

[28] Morrison, D.L., *et al.*, "Speech-Controlled Text-Editing: Effects of Input Modality and of Command Structure," *Int. Journal of Man-Machine Studies*, 1984, **21**(1), pp. 49-63.

[29] Nielsen, J., *Usability Engineering*, 1993, Boston, MA, Academic Press.

[30] Norman, D.A., *Things That Make Us Smart: Defending Human Attributes in the Age of the Machine*, 1993, Reading, MA, Addison-Wesley.

[31] O'Shaughnessy, D., "Specifying Accent Marks in French Text for Teletext and Speech Synthesis," *Int. Journal of Man-Machine Studies*, 1989, **31**(4), pp. 405-414.

[32] Pierrehumbert, J., and M. Beckman, *Japanese Tone Structure*, 1988, Cambridge, MA, MIT Press.

[33] Poock, G.K., "Voice Recognition Boosts Command Terminal Throughput," *Speech Technology*, 1982, **1**, pp. 36-39.

[34] Rao, P.V.S., "VOICE: An Integrated Speech Recognition Synthesis System for the Hindi Language," *Speech Communication*, 1993, **13**, pp. 197-205.

[35] Royer, T., "Using Scenario-Based Designs to Review User Interface Changes and Enhancements," *Proc. of DIS'95 Symposium on Designing Interactive Systems: Processes, Practices, Methods, & Techniques*, 1995, pp. 237-246.

[36] Rubin, J., *Handbook of Usability Testing: How to Plan, Design, and Conduct Effective Tests*, 1994, New York, John Wiley.

[37] Sagisaka, Y. and L.S. Lee, "Speech Recognition of Asian Languages" in *Proc. IEEE Automatic Speech Recognition Workshop* 1995, Snowbird, UT, pp. 55-57.

[38] Salvendy, G., *Handbook of Human Factors and Ergonomics*, 1997, New York, John Wiley.

[39] Schmandt, C., *Voice Communication with Computers*, 1994, New York, Van Nostrand Reinhold.

[40] Schmidt, M., *et al.*, "Phonetic Transcription Standards for European Names (ONOMASTICA)," *Proc. of the European Conf. on Speech Communication and Technology*, 1993, Berlin, pp. 279-283.

[41] Shneiderman, B., *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 1997, Reading, MA, Addison-Wesley.

[42] Spencer, R.H., *Computer Usability Testing and Evaluation*, 1985, Englewood Cliffs, NJ, Prentice-Hall.

[43] Weinschenk, S. and D. Barker, *Designing Effective Speech Interfaces*, 2000, New York, John Wiley.

[44] Wixon, D. and J. Ramey, *Field Methods Casebook for Software Design*, 1996, New York, John Wiley.

[45] Wood, L., *User Interface Design: Bridging the Gap from User Requirements to Design*, 1997, CRC Press.

[46] Yang, K.C., *et al.*, "Statistics-Based Segment Pattern Lexicon—A New Direction for Chinese Language Modeling," *Int. Conf. on Acoustics, Speech and Signal Processing*, 1998, pp. 169-172.

# INDEX


A
A* search, 603, 606, 626-39
  admissible heuristics, 630-31
  extending new words, 631-34
  fast match, 634-38
  multistack search, 639
  stack decoders, 592
  stack pruning, 638-39
Abbreviations, 709-12
  disambiguation, 711
  expansion, 711-12
Absolute Category Rating (ACR), 841
Absolute discounting, 568
Absolute threshold of hearing, 23, 36
Abstract prosody, 745-61
  accent, 751-53
  pauses, 747-49
  prosodic phrases, 749-51
  prosodic transcription systems, 759-61
    INTSINT, 760
    PROSPA, 759
    TILT, 760
  tone, 753-57
  tune, 757-59
Accent, 751-53
Accessibility, 929
Acoustical environment, 477, 478-86
  additive noise, 478-80
  babble noise, 479
  cocktail party effect, 479
  Lombard effect, 480
  model of the environment, 482-86
  pink noise, 478
  reverberation, 480-82
  white noise, 478-79
Acoustical model of speech production, 283-90
  glottal excitation, 284
  lossless tube concatenation, 284-88
  mixed excitation model, 289
  source-filter models, 288-90
Acoustical transducers, 486-97
  active microphones, 496
  bidirectional microphones, 490-94
  carbon button microphones, 497
  condenser microphone, 486-89
  directionality patterns, 489-96
  dynamic microphones, 497
  electromagnetic microphones, 497
  electrostatic microphones, 497
  passive microphones, 496
  piezoelectric microphones, 497
  piezoresistive microphones, 497
  pressure gradient microphones, 496
  pressure microphones, 496
  ribbon microphones, 497
  unidirectional microphones, 494-96
Acoustic modeling, 415-75
  adaptive techniques, 444-53
    clustered models, 452-53
    maximum likelihood linear regression (MLLR), 447-50
    maximum a posteriori (MAP), 445-47
    MLLR vs. MAP, 450-51
  confidence models, 453-57
    filler models, 453-54
    transformation models, 454-55
  historical perspective, 465-68
  neural networks, 457-59
    integrating with HMMs, 458-59
    parametric trajectory models, 460-62
    recurrent, 457-58
    time delay neural network (TDNN), 458
  scoring acoustic features, 439-43
    HMM output distributions, 439-41
    isolated vs. continuous speech training, 441-43
  speech signals:
    context variability, 417
    environment variability, 419
    speaker variability, 418-19
    style variability, 418
  word recognition errors, types of, 420
Acoustic pattern matching, 545
Acronyms, 711-12
Active arcs, 550
Active constituents, 550-51
Active microphones, 496
Adaptive codebook, 356-57
Adaptive delta modulation (ADM), 347
Adaptive echo cancellation (AEC), 497-504
  LMS algorithm, 499-500
  normalized LMS algorithm (NLMS), 501-2
  RLS algorithm, 503-4
  transform-domain LMS algorithm, 502-3
Adaptive language models, 575-78
  cache language models, 574-75
  maximum entropy models, 576-78
  topic-adaptive models, 575-76
Adaptive PCM, 344-45
Adaptive spectral entropy coding (ASPEC), 350-51
Additive noise, 478-80
Adjectives, 54
ADPCM, 348
Adverbs, 60
AEC, *See* Adaptive echo cancellation (AEC)
Affricates, 44
Agent-based dialog modeling, 914
Air Travel Information Service (ATIS) task, 901-3
A-law compander, 343

Algebraic code books, 358-59
Algorithmic delay, 339
Aliasing, 245
Allophones, 47-49
Alphabet, 121
Alternative hypothesis, 114
Alternatives question, 62
Alveolar consonants, 46
American Institute of Electrical Engineers (AIEE), 272
American National Standards Institute (ANSI), 360
Amplitude modulator, 208
Analog signals, 202
  analog-to-digital conversion, 245-46
  digital processing of, 242-48
  digital-to-analog conversion, 246-48
  Fourier transform of, 243
  sampling theorem, 243-45
Analysis by synthesis, 353-56
Analysis frames, 276
Anaphora, 882
Anaphora resolution, 883-84
Anechoic chamber, 480
Anger, and speech, 745
Anti-causal system, 211
Anti-jam (AJ), 361
Application programming interface (API), 920-21
Applications:
  accessibility, 929
  automobile, 930
  classes of, 921-31
  computer command and control, 921-24
  dictation, 926-29
  handheld devices, 930
  hands-busy, eyes-busy, 927
  speaker recognition, 931
  telephony applications, 924-26
Approximants, 42
Articulation, of English consonants, 42, 45
Articulators, 24-25
Articulatory speech synthesis, 793, 803-4, 846
Artificial intelligence (AI), 855, 889
Attentional state, 859
Audio coding, 338
Aurora project, 925
Autocorrelation method, 324-27
Automobile applications, 930
AutoPC (Clarion), 930
Auto-regressive (AR) modeling, 290

## B

Babble noise, 479
Back-channel turn, 861
Backoff models, 563-64
Backoff paths, 618-19
Backoff smoothing, 565-70
Back propagation algorithm, 163
Backtracking, 598
Backus-Nauer form (BNF), 69
Backward prediction error, 297
Backward reasoning, 595
Bandlimited interpolation, 245
Bandpass filter, 242
Bark scale, 32-33
Bark scale functions, 35
Basic search algorithms, 591-643
  blind graph, 597-601
    breadth-first, 600-601
    depth-first, 598-99
    heuristic graph, 601-8
  complexity, 592
  general graph searching procedures, 593-97
  graph-search algorithm, 597
  historical perspective, 640
  search algorithms for speech recognition, 608-12
  stack decoding (A* search), 626-39
    admissible heuristics, 630-31
    extending new words, 631-34
    fast match, 634-38
    multistack search, 639
    stack pruning, 638-39
  time-synchronous Viterbi beam search, 622-26
    use of beam, 624-25
    Viterbi beam search, 625-26
Baum-Welch algorithm, 389-93
Bayes' classifiers:
  comparing, 148-49
  representation, 138-39
Bayes' decision theory, 133, 134-49, 159
  curse of dimensionality, 144-46
  discriminant functions, 138-41
  error rate, estimating, 146-48
  Gaussian classifiers, 142-44
  minimum-error-rate decision rules, 135-38
Bayes' estimator, 99
Bayesian estimation, 107-13
  general, 109-10
  prior and posterior estimation, 108-9
Bayes' risk, 136
Bayes' rule, 75-78
Bayes' theorem, 128
Beamforming, 507
Beam search, 606-8
Behavior model, 855
Best-first search, 602-6
Bidirectional microphones, 490-94
Bidirectional search, 595-96
Bigrams, 559, 563
  search space with, 617-18
Bilinear transforms, 315-16
Binaural listening, 31
Binomial distributions, 86
Bit, 121
Bit reversal, 224
Blind graph search algorithms, 597-601

Blind source separation (BSS), 510-15
Block coding, 125
Bolzmann's constant, 488
Bottom-up chart parsing, 550-53
Bound stress, 431
Branch-and-bound algorithm, 604, 640
Branching factor, 593-95
Breadth-first search, 600-601, 624
Breathy-voiced speech, 330
Bridging, 488
Brute-force search, 601

C
Cache language models, 574-75, 882
Carbon button microphones, 497
Cardioid mikes, 495
CART, *See* Classification and regression trees (CART)
CART-based durations, 763
Cascade/parallel formant synthesizer, 797-802
  parameter values, 799
  targets, 801-2
Case relations, 63, 66
Cauchy-Schwarz inequality, 263
CCITT (Comite Consultatif International Telephonique et Telegraphique), 343
CELP, *See* Code excited linear prediction (CELP)
Central Limit Theorem, 93
Centroid, 165
Cepstral mean normalization (CMN), 522-24, 540
Cepstrum, 306-15
  cepstrum vector, 309
  complex, 307-8
  LPC-cepstrum, 309-11
  of periodic signals, 311-12
  of pole-zero filters, 308-98
  real, 307-8
  source-filter separation via, 314-15
  of speech signals, 312-13
Chain rule, 75, 77-78, 124
Channel coding, 126-28
Character, and prosody, 744
Chart, 550
Chart parsing for context-free grammars, 549-53
  bottom-up, 550-53
  top down vs. bottom up, 549-50
CHATR system of ATR, 781
Chebychev polynomials, 237
Chemical formulae, 719
Chemical Markup Language (CML), 719
Chomsky hierarchy, 547-48
Chunking, 876
Chunks, 691
Circular convolution, 227
Class-conditional pdfs, 133, 140, 144
Classification and regression trees (CART), 134, 175-89, 191, 729-30, 748, 879
  CART algorithm, 189
  choice of question set, 177-78
  complex questions, 182-84
  growing the tree, 181
  missing values and conflict resolution, 182
  right-sized tree, 184-89
    cross-validation, 188-89
    independent test sample estimation, 187-88
    minimum cost-complexity pruning, 185-87
  splitting criteria, 178-81
Class inclusion, 66
Class *n*-grams, 570-74
  data-driven classes, 572-73
  rule-based classes, 571-72
Clauses, 61-62
  relative, 61
Clear /l/, 48
Cleft sentence, 62
Closed-loop estimation, 356
Closed-phase analysis, 319
Closed POS categories, 54
Cluster, 572
Clustered acoustic-phonetic units, 432-36
Clustered models, 452-53
CMU Pronunciation Dictionary, 436
Coarticulation, 47, 49-51
Cochlea, 30
Cocke-Younger-Kasmi (CYK) algorithm, 584
Cocktail party effect, 479
Codebook, 164-65
Code Division Multiple Access (CDMA), 360-61
Code excited linear prediction (CELP), 353-61
  adaptive codebook, 356-57
  analysis by synthesis, 353-56
  LPC vocoder, 353
  parameter quantization, 358-59
  perceptual weighting/postfiltering, 357-58
  pitch prediction, 356-57
  standards, 359-61
Coder delay, 339
Codeword, 164-65
Colored noise, 270
Combination models, 456-57
COMLEX dictionary (LDC), 436-37
Command and control speech recognition, 921-24
  application ideas/uses, 923
  situations for, 922-23
Commissives, 861
Communicative prosody, 858
Compact Disc-Digital Audio (CD-DA), 342
Compander, 342
Comparison Category Rating (CCR) method, 842
Complements, 58, 59
Complex cepstrum, 307-8
Complexity parameter, 185
Compressions, 21
Computational delay, 339

Concatenative speech synthesis, 793-94
choice of unit, 805-8
context-dependent phonemes, 808-9
context-independent phonemes, 806-7
diphones, 807-8
with no waveform modification, 794
optimal unit string, 810-17
data-driven transition cost, 815-16
data-driven unit cost, 816-17
empirical transition cost, 812-13
empirical unit cost, 813-15
objective function, 810-12
subphonetic units (senones), 809
syllables, 809
unit inventory design, 817-18
with waveform modification, 795
word and phrase, 809
Concept-to-speech rendition, 899-901
Conceptual graphs, 872-73
Condenser microphone, 486-89
Conditional entropy, 123-24
Conditional expectation, 81
Conditional likelihood, 151
Conditional maximum likelihood estimator (CMLE), 151
Conditional probability, 75-76
Conditional risks, 136
Conditioning, 320
Conference of European Posts and Telegraphs (CEPT), 360
Confidence models, 453-57
combination models, 456-57
filler models, 453-54
transformation models, 454-55
Conflict resolution procedure, 182
Conjugate quadrature filters, 251-54
Conjunctions, 54
Connotation, message, 739
Consonants, 24, 42-46
affricates, 44
alveolar, 46
dental, 46
fricatives, 42, 44
labial, 46
labio-dental, 46
nasal, 43-44
obstruent, 43
palatal, 46
plosive, 42-43
stop, 43
velar, 46
Consumer audio, 351-52
Content words, 54
Context coarticulation, 735
Context dependency, 430-31
Context-dependent phonemes, 808-9
Context-dependent units and inter-word triphones, 658-59
Context-free grammar (CFG), 465, 547, 921
vs. *n*-gram models, 580-84
search space, 613-16
Context-free grammars (CFGs), search architecture, 676-77
Context-independent phonemes, 806-7
Context variability, 417
Continuation rise, 749
Continuous distribution, 78
Continuous-frequency transforms, 209-16
Fourier transforms, 208-10
*z*-transforms, 211-15
Continuously listening model, 422
Continuously variable slope delta modulation (CVSDM), 347
Continuous mixture density HMMs, 394-96
Continuous random variable, 78
Continuous speech recognition (CSR), 591, 611-12, 945
Continuous speech training, vs. isolated speech training, 441-43
Continuous-time stochastic processes, 260
Contrastive stress, 431
Contrasts, 66
Convolution operator, 207
Co-references, 882
*Corpora*, 545-46
Corpus-based F0 generation, 779-82
F0 contours indexed by parsed text, 779-81
F0 contours indexed by ToBI, 781-82
transplanted prosody, 779
Corrective training, 158
Correlation, 82-83
Correlation coefficient, 82-83
Covariance, 82-83
Covariance matrix, 84
Critical region, 114
Cross-validation, 188-89
Cumulative distribution function, 79
Currency, 716
Curse of dimensionality, 144-46

## D

DAMSL system, *See* Dialog Act Markup in Several Layers (DAMSL)
Dark /l/, 48
DARPA, 11
DARPA ATIS programs, 913
Data-directed search, 549
Data-driven parallel model combination (DPMC), 533
Data-driven speech synthesis, 794, 803
Data flow, 694-97
DAVO, 846
DCT, *See* Discrete Cosine Transform (DCT)
Decibels (dB), 22

Decimation-in-frequency, 223
Decimation-in-time, 223
Declaratives, 861
Declarative sentence, 62
Declination, 766-67
Decoder, 5
Decoder basics, 609
Decoder delay. 339
DECTalk system, 736, 846
Degree of displacement, 21-22
Deixis, 882
Deleted interpolation smoothing, 564-65
Deletion errors, 420
Delta modulation (DM), 346
Denotation, message, 739
Dental consonants, 46
Depth-first search, 598-99
Derivational morphology, 56-57
Derivational prefixes, 57
Derivational suffixes, 57
Descrambling, 224
Deterministic signals, 260
Development set, 419-20
Diagnostic Rhyme Test (DRT), 837
Dialects, 725
Dialog Act Markup in Several Layers (DAMSL), 862-66
Dialog-act theory, 914
Dialog control, 866-67, 949
Dialog flow, 942-43
  prompting strategy, 943
  prosody, 943
  spoken menus, 942
Dialog grammars, 887-88
Dialog management, 886-94
  dialog grammars, 887-88
  plan-based systems, 888-92
Dialog management module, 881
Dialog Manager, 7-8, 855
Dialog repair, 885
Dialog (speech) acts, 861-66
Dialog structure, 859-67
  attentional state, 859
  Dialog Act Markup in Several Layers (DAMSL), 862-66
  intentional state, 859
  linguistic forms, 859
  task knowledge, 859
  units of dialog, 860-61
  world knowledge, 859
Dialog system, 854-55
  dialog manager, 855
  discourse analysis, 855
  semantic parser, 854-55
Dialog turns, 705-6
Dictation, 926-29, 935, 948
Dielectric, 486
Differential pulse code modulation (DPCM), 345-48
Differential quantization, 345-48
Digital Audio Broadcasting (DAB), 352
Digital filters and windows, 229-42
  generalized Hamming window, 231-32
  ideal low-pass filter, 229-30
  rectangular window, 230-31
  window functions, 230-32
Digital signal processing, 201-73
  of analog signals, 242-48
    analog-to-digital conversion, 245-46
    digital-to-analog conversion, 246-48
    Fourier transform of, 243
    sampling theorem, 243-45
  circular convolution, 227
  continuous-frequency transforms, 209-16
    of elementary functions, 212-15
    Fourier transform, 208-10
    properties, 215-16
    z-transforms, 211-12
  digital filters and windows, 229-42
    generalized Hamming window, 231-32
    ideal low-pass filter, 229-30
    rectangular window, 230-31
    window functions, 230-32
  digital signals/systems, 202-8
  Discrete Cosine Transform (DCT), 228-29
  discrete-frequency transforms, 216-29
    discrete Fourier transform (DFT), 218-19
    Fourier transforms of periodic signals, 219-22
  Fast Fourier Transforms (FFT), 222-27
    FFT subroutines, 224-27
    prime-factor algorithm, 224
    radix-2 FFT, 222-23
    radix-4 algorithm, 223
    radix-6 algorithm, 223
    radix-8 algorithm, 223
    split-radix algorithm, 223
  filterbanks, 251-60
    DFTs as, 255-58
    multiresolution, 254-55
    two-band conjugate quadrature filters, 251-54
  FIR filters, 229, 232-38
    first-order, 234-35
    linear-phase, 233-34
    Parks McClellan algorithm, 236-38
    window design FIR lowpass filters, 235-36
  IIR filters, 238-42
    first-order, 239-41
    second-order, 241-42
  multirate signal processing, 248-51
    decimation, 248-49
    interpolation, 249-50
    resampling, 250-51
  stochastic processes, 260-70
    continuous-time, 260
    discrete-time, 260

Digital signal processing, stochastic processes *(cont.)*
LTI systems with stochastic inputs, 267
noise, 269-70
power spectral density, 268-69
stationary processes, 264-67
statistics of, 261-64
Digital Signal Processing (DSP), 202-3, 339
Digital signals/systems, 202-8
digital systems, 206-8
linear time-invariant (LTI) systems, 207
linear time-varying systems, 208
nonlinear systems, 208
other digital signals, 206
sinusoidal systems, 203-5
Digital systems, defined, 202
Digital-to-analog conversion, 246-48
Digital wireless telephony applications, 925
Diphones, 807-8
Diphthongs, 40
Directionality patterns, 489-96
Directives, 861
Disambiguation, 876
Discourse analysis, 7, 753, 855, 881-86
resolution, 882-85
Discourse memories, 882
Discourse segments, 857
Discrete Cosine Transform (DCT), 228-29
Discrete distribution, 77
Discrete-frequency transforms, 216-29
discrete Fourier transform (DFT), 218-19
Fourier transforms of periodic signals, 219-22
Discrete joint distribution, 83-84
Discrete random variables, 77
Discrete-time Fourier transform, 209, 210
Discrete-time stochastic processes, 260
Discriminative training, 150-63
gradient descent, 153-55
maximum mutual information estimation (MMIE), 150-52
minimum-error-rate estimation, 156-58
multi-layer perceptrons, 160-63
neural networks, 158
single-layer perceptrons, 159-60
Disfluency, 857
Distortion measures, 164-66
Distribution function, 79
Document structure detection, 692, 699-706
chapter and section headers, 700-701
dialog turns and speech acts, 705-6
email, 704-5
lists, 701-2
paragraphs, 702
sentences, 702-4
Web pages, 705
Dolby Digital, 351
Domain knowledge, 2
Domain-specific tags, 718-20
chemical formulae, 719
mathematical expressions, 718-19
miscellaneous formats, 719-20
Dragon NaturallySpeaking, 926
Dr. Who case study, 906-13
dialog manager, 910-13
discourse analysis, 909-10
semantic parser (sentence interpolation), 908
semantic representation, 906-8
Dr. Who Project, 869, 876-77
DTS, 351-52
Duration assignment, 761-63
CART-based durations, 763
rule-based methods, 762-63
Dynamic microphones, 497
Dynamic time warping (DTW), 383-85

## E

Ear:
cochlea, 30
eardrum, 29
middle ear, 29
outer ear, 29
oval window, 29
physiology of, 29-32
sensitivity of, 30
Eardrum, 29
Earley algorithm, 584
Eigensignals, 209
Eigenvalue, 209
Electret microphones, 487
Electroglottograph (EGG), 828
signals, 319-20
Electromagnetic microphones, 497
Electronic Industries Alliance (EIA), 360
Electrostatic microphones, 497
Ellipsis, 882
EM algorithm, 134, 170-72
Embedded ADPCM, 348
Emotion, and prosody, 744-45
Emphatic stress, 431
End-point detection, 422-24
Entropy, 120-22
conditional, 123-24
Entropy coding, 350-51
Environmental model adaptation, 528-38
model adaptation, 530-31
parallel model combination, 531-34
retraining on compensated features, 537-38
retraining on corrupted speech, 528-39
vector Taylor series, 535-37
Environmental robustness, 477-544
acoustical environment, 477, 478-86
additive noise, 478-80
babble noise, 479
cocktail party effect, 479

Lombard effect, 480
model of the environment, 482-86
pink noise, 478
reverberation, 480-82
white noise, 478-79
acoustical transducers, 486-97
active microphones, 496
bidirectional microphones, 490-94
carbon button microphones, 497
condenser microphone, 486-89
directionality patterns, 489-96
dynamic microphones, 497
electromagnetic microphones, 497
electrostatic microphones, 497
passive microphones, 496
piezoelectric microphones, 497
piezoresistive microphones, 497
pressure gradient microphones, 496
pressure microphones, 496
ribbon microphones, 497
unidirectional microphones, 494-96
adaptive echo cancellation (AEC), 497-504
convergence properties of the LMS algorithm, 500-501
LMS algorithm, 499-500
normalized LMS algorithm (NLMS), 501-2
RLS algorithm, 503-4
transform-domain LMS algorithm, 502-3
environmental model adaptation, 528-38
model adaptation, 530-31
parallel model combination, 531-34
retraining on compensated features, 537-38
retraining on corrupted speech, 528-39
vector Taylor series, 535-37
environment compensation preprocessing, 515-28
cepstral mean normalization (CMN), 522-24
frequency-domain MMSE from stereo data, 519-20
real-time cepstral normalization, 525
spectral subtraction, 516-19
use of Gaussian mixture models, 525-28
Weiner filtering, 520-22
multimicrophone speech enhancement, 504-15
blind source separation (BSS), 510-15
microphone arrays, 505-10
nonstationary noise, modeling, 538-39
Environment compensation preprocessing, 515-28
cepstral mean normalization (CMN), 522-24
frequency-domain MMSE from stereo data, 519-20
real-time cepstral normalization, 525
spectral subtraction, 516-19
use of Gaussian mixture models, 525-28
Wiener filtering, 520-22
Environment variability, 419
Epoch detection, 828-29
Equal-loudness curves, 31
Ergodic processes, 265-67
Ergonomics of Software User Interface, 932
Error handling, 937-41
error detection and correction, 938-39
feedback and confirmation, 939-41
Estimation, 98-99
Estimation theory, 98-113
Bayesian estimation, 107-13
general, 109-10
prior and posterior estimation, 108-9
least squared error (LSE) estimation, 99-100
for constant functions, 100
for linear functions, 101-2
for nonlinear functions, 102-4
MAP estimation, 111-13
maximum likelihood estimation (MLE), 104-7
minimum mean squared error (MMSE), 99-104
for constant functions, 100
for linear functions, 101-2
for nonlinear functions, 102-4
Euclidean distortion measure, 165-66
Eureka 147 DAB specification, 352
European Telecommunication Standards Institute (ETSI), 360
Evaluation of understanding and dialog, 901-3
and ATIS task, 901-3
PARADISE framework, 903-6
Exact $n$-best algorithm, 666-67
Exception list, 697, 728
Excitation signal, 301
Exclamative sentence, 62
Exhaustive search, 597
Expectation (mean) vector, 84
Expectation of a random variable, 79
Exponential distribution, 98


## F


F0 contour interpolation, 772-73
F0 jumps, 330
F1/F2 targets, 39
Factored language probabilities, 650-53
Factored lexical trees, 652-53
Fast Fourier Transforms (FFT), 222-27
FFT subroutines, 224-27
prime-factor algorithm, 224
radix-2 FFT, 222, 223
radix-4 algorithm, 223
radix-6 algorithm, 223
radix-8 algorithm, 223
split-radix algorithm, 223
Fast match, 634-38, 661-62
look-ahead strategy, 661-62
Rich-Get-Richer (RGR) strategy, 662
Fear, and speech, 745
Feedback, 490
Feedforward adaptation, 345
Fenones, 467
Festival, 732-35
FFT, *See* Fast Fourier Transforms (FFT)

Filler models, 453-54
Filterbanks, 251-60
  DFTs as, 255-58
  multiresolution, 254-55
  two-band conjugate quadrature filters, 251-54
Filters, 210
Finite-impulse response (FIR), *See* FIR filters
Finite-state automaton, 547
Finite-state grammar, 613-16
Finite-state machines (FSM), 548
Finite state network, 654
FIR filters, 229, 232-38
  first-order, 234-35
  linear-phase, 233-34
  Parks McClellan algorithm, 236-38
  window design, 235-36
First coding theorem, 124
First-order FIR filters, 234-35
First-order IIR filters, 239-41
First-order moment, 261
Focus, 883
Focus shifts, cueing, 892
Formal language modeling, 546-53
  chart parsing for context-free grammars, 549-53
    bottom-up, 550-53
    top down vs. bottom up, 549-50
  Chomsky hierarchy, 547-48
Formant frequencies, 319-23
  statistical formant tracking, 320-23
Formants, 27-28
Formant speech synthesis, 793, 796-804
  cascade model, 797
  formant generation by rule, 800-803
  Klatt's cascade/parallel formant synthesizer, 797-99
  locus theory of speech production, 800
  parallel model, 797
  waveform generation from formant values, 797-99
Formant targets, 39
Forward algorithm, 385-87
Forward-backward algorithm, 389-93, 442-43, 557
Forward-backward search algorithm, 670-73
Forward error correction (FEC), 352
Forward prediction error, 297
Forward reasoning, 595
Fourier series expansion, 218
Fourier transforms, 208-10
  Fast Fourier Transforms (FFT), 222-27
    FFT subroutines, 224-27
    prime-factor algorithm, 224
    radix-2 FFT, 222-23
    radix-4 algorithm, 223
    radix-6 algorithm, 223
    radix-8 algorithm, 223
    split-radix algorithm, 223
  properties of, 215-17
Fourier transforms of periodic signals, 219-22
  complex exponential, 219-20
  general periodic signals, 221-22
  impulse train, 221
Frames, 339
Free stress, 431
Frequency analysis, 32-34
Frequency domain, advantages of, 348-49
Frequency-domain MMSE from stereo data, 519-20
Frequency masking, *See* Masking
Frequency response, 210
Fricatives, 42, 44
Fujisaki's model, 776
Full duplex sound cards, 936
Functionality encapsulation, 871-72
Functional tests, 842-43
Function words, 54
Fundamental frequency, 25

## G

G.711 standard, 343-44, 348, 359, 371
G.722 standard, 348, 359
G.723.1 standard, 359
G.727 standard, 348
G.728 standard, 359
G.729 standard, 359
Game search, 594
Gamma distributions, 90-91, 95
Gaussian distributions, 92-98
  Central Limit Theorem, 93
  lognormal distribution, 97-98
  multivariate mixture Gaussian distributions, 93-95
  standard, 92-93
  $\chi^2$ distributions, 95-96
Gaussian mixture models, 525-28
Gaussian processes, 264-65
General graph searching procedures, 593-97
Generality, of grammar, 546
Generalized Hamming window, 231-32
Generalized Lloyd algorithm, 168
Generalized triphones, 808
General Packet Radio Service (GPRS), 361
Geometric distributions, 86-87
Gibbs phenomenon, 235
Gini index of diversity, 181
Glides, 42
Global Positioning System (GPS), 868, 930
Glottal cycle, 26
Glottal excitation, 284
Glottal stop, 43
Glottis, 25, 288
Glyphs, 36
Goal-directed search, 549
Goodness-of-fit test, 116-18
Good-Turing estimates and Katz smoothing, 565-67
Gradient descent, 153-55, 190
Gradient prominence, 765-66
Graham-Harison-Ruzzo algorithm, 584
Grammar, 545

Granular noise, 346
Grapheme-to-phoneme conversion, 692-93
Graphical user interface (GUI), 1
Graph-search algorithms, 591, 597
Greedy symbols, 558
Ground, 900
Grounding, 861

H
H.323, 359
Half duplex sound cards, 936
Half phone, 809
Hamming window, 232, 258, 278-80, 283
  generalized, 231-32
Handheld devices, 930
Hands-busy, eyes-busy applications, 927
Hanning window, *See* Hamming window
Hard palate, 25
Harmonic coding, 363-67
  parameter estimation, 364-65
  parameter quantization, 366-67
  phase modeling, 365-66
Harmonic errors, 330
Harmonic sinusoids, 218
Harmonic/Stochastic (H/S) model, 847
Harvard Psychoacoustic Sentences, 839
*Has-a* relations, 65
Haskins Syntactic Sentence Test, 839
Head-noun, 59
Head of a phrase, 59
Headset microphone, 936
Hearing sensitivity, 30
Hermitian function, 265
Hertz (Hz), 21
Hessian of the least-squares function, 504
Hessian matrix, 154
Heuristic graph search, 601-8
  beam search, 606-8
  best-first (A* search), 602-6
Heuristic information, 601-2
Heuristic search methods, 601
Hidden Markov models (HMM), 56, 134, 170, 377-413, 416, 547, 931
  Baum-Welch algorithm, 389-93
  continuous mixture density, 394-96
  decoding, 387-89
  definition of, 380-93
  deleted interpolation, 401-3
  dynamic programming, 384-85
    advantage of, 384
    algorithm, 385
  dynamic time warping (DTW), 383-85
  estimating parameters, 389-93
  evaluating, 385-87
  forward algorithm, 385-87
  forward-backward algorithm, 389-93
  historical perspective, 409-10
  initial estimates, 398-99
  limitations of, 405-9
    conditional independence assumption, 409
    duration modeling, 406-8
    first-order assumption, 408
  Markov chain, 378-80
    Markov assumption for, 382
    output-independence assumption, 382
  model topology, 399-401
  observable Markov model, 379-80
  parameter smoothing, 403-4
  practical issues, 398-405
  probability representations, 404-5
  semicontinuous, 396-98
  training criteria, 401
  Viterbi algorithm, 387-89
Hidden understanding model (HUM), 879-80
High-frequency sounds, 31
High-pass filters, 235
Hill-climbing style of guidance, 601
H method, 147
HMM, *See* Hidden Markov models (HMM)
Holdout method, 147
Home applications, 921
Homograph disambiguation, 693, 723, 724-25
Homographs, 721
Homomorphic transformation, 306, 312
Huffman coding, 125-26
Human Factors and Ergonomic Society (HFES), 931-32
Human-machine interaction, 1

I
Ideal low-pass filter, 229-30
IIR filters, 238-42
  first-order, 239-41
  second-order, 241-42
Imperative sentence, 62
Implicit confirmation, 892-93
Implicit memory, 882-83
Impulse response, 207
Inconsistency checking, 885-86
Inconsistency detection, 881
Independent component analysis (ICA), 510
Independent identically distributed (iid), 82
Independent processes, 264
Independent test sample estimation, 187-88
Indistinguishable states, 654
Infinite-impulse response (IIR) filters, *See* IIR filters
Inflectional morphology, 56
Inflectional suffix, 57
Infomax rule, 512-13
Information theory, 73-131
  channel coding, 126-28
  conditional entropy, 123-24
  entropy, 120-22
  mutual information, 126-28

Information theory *(cont.)*
origin of, 74
source coding theorem, 124-26
Informed search, 604
Infovox TTS system, 846-47
Inner ear, 29
Input method editors (IME), 736
Insertion errors, 420
Insertion penalty, 610
Inside constituent probability, 555
Inside-outside algorithm, 555
Instance definition, 868
Instantaneous coding, 125
Instantaneous mixing, 510
Instantaneous mutual information, 151
Institute of Electrical and Electronic Engineers (IEEE), 272
Institute of Radio Engineers (IRE), 272
Intelligibility tests, 837-39
Intentional state, 859
Interactive voice response (IVR) systems, 924
Intermediate phrase break, 751
International Conference on Acoustic, Speech and Signal Processing (ICASSP), 272
Internationalization, 943-45
International Telecommunication Union (ITU), 343
International Telecommunication Union-Radiocommunication (ITU-R), 352
Interpolated models, 564-65
Interword-context-dependent phones, 430
Inter-word triphones, 658-59
Intonational phrase break, 751
Intonational phrases, 53, 749
INTSINT, 760
Inverse filter, 290
Inverse-square-law effect, 494
Inverse $z$-transform, 212
of rational functions, 213-15
*Is-a* taxonomies, 64-66
Isolated vs. continuous speech training, 441-43
Isolated word recognition, 610-11

## J

Japanese vowels, 46-47
Jensen's inequality, 122
Jitter, 768
Joint distribution function, 84
Jointly strict-sense stationary, 264
Joint probability, 74
Joy, and speech, 745
JSAPI, 921
Juncture, 746-47
Just noticeable distortion (JND), 35

## K

Kalman filter, 522
Karhunen-Loeve transform, 426
Katz' backoff mechanism, 618
Katz smoothing, 565-67
Klattalk system, 846
Klatt's cascade/parallel formant synthesizer, 797-802
parameter values for, 799
targets used in, 801-2
$K$-means algorithm, 166-69
Kneser-Ney smoothing, 568-70, 573
Knowledge sources (KSs), 646, 663, 673-74
Kolmogorov-Smirnov test, 118
Kronecker delta, 220
$K$th moment, 80
Kullback-Leibler (KL) distance, 122, 581

## L

Labial consonants, 46
Labio-dental consonants, 46
Lancaster/IBM Spoken English Corpus, 751-52
Language modeling, 545-90
adaptive, 575-78
cache language models, 574-75
maximum entropy models, 576-78
topic-adaptive models, 575-76
CFG vs. $n$-gram models, 580-84
complexity measure of, 560-62
formal, 546-53
chart parsing for context-free grammars, 549-53
Chomsky hierarchy, 547-48
historical perspective, 584
$n$-gram pruning, 580-81
$n$-gram smoothing, 562-74
backoff smoothing, 565-70
class $n$-grams, 570-74
deleted interpolation smoothing, 564-65
performance of, 573-74
stochastic language models, 554-60
$n$-gram language models, 558-60
probabilistic context-free grammars, 554-58
vocabulary selection, 578-80
Language model probability, 610
Language models, 4, 949
Language model states, 613-22
backoff paths, 618-19
search space:
with bigrams, 617-18
with FSM and CFG, 613-16
with trigrams, 619-20
with the unigram, 616-17
silences between words, 621-22
Lapped Orthogonal Transform (LOT), 260
Large-vocabulary search algorithms, 645-85
context-dependent units and inter-word triphones, 658-59
exact $n$-best algorithm, 666-67
factored language probabilities, 650-53
factored lexical trees, 652-53
finite state network, 654

forward-backward search algorithm, 670-73
historical perspective, 681-82
HMM:
different layers of beams, 660-61
fast match, 661-62
lexical successor trees, 652
lexical trees, 646-48
handling multiple linguistic contexts in, 657-58
linear tail in, 655
optimization of, 653
lexical tree search, 648
Microsoft Whisper, 676-81
CFG search architecture, 676-77
*n*-gram search architecture, 677-81
*n*-best and multipass search strategies, 663-74
one-pass *n*-best and word-lattice algorithm, 669-70
one-pass vs. multipass search, 673-74
polymorphic linguistic context assignment, 656-57
prefix trees, 647
pronunciation trees, multiple copies of, 648-50
search-algorithm evaluation, 674-76
sharing tails, 655-56
single-word subpath, 655
subtree dominance, 656
subtree isomorphism, 654
subtree polymorphism, 656-58
tree lexicon, 646-59
word-dependent *n*-best and word-lattice algorithm, 667-70
word-lattice generation, 672-73
Larnyx, 25
vocal fold cycling at, 26
Laryngograph, 828
Lateralization, 31
Lateral liquid, 42
Law of large numbers, 82
LBG algorithm, 169-70
Least squared error (LSE) estimation, 99-100
for constant functions, 100
for linear functions, 101-2
for nonlinear functions, 102-4
Least squared regression methods, 180
Least square error (LSE), 160
Leave-one-out method, 147
Left-recursive grammar, 550
Lempel-Ziv coding, 126
Lernout&Hauspie's Voice Xpress, 926
Letter-to-sound (LTS) conversion, 437, 693, 728-30
Letter-to-sound (LTS) rules, 697
Level of significance, 114-15
Levinson-Durbin recursion, 297-98, 333
Lexical baseforms, 436-39
Lexical knowledge, 545
Lexical part-of-speech (POS), 53-56
Lexical successor trees, 652
Lexical trees, 646-48
handling multiple linguistic contexts in, 657-58
linear tail in, 655
optimization of, 653
Lexicon, 697-98
Light ///, 48
Likelihood function, 104
Likelihood ratio, 139
Limited-domain waveform concatenation, 794
Linear bounded automaton, 548
Linear Discriminate Analysis (LDA), 427
Linear-phase FIR filters, 233-34
Linear predictive coding (LPC), 290-306
autocorrelation method, 295-96
covariance method, 293-94
equivalent representations, 303-6
lattice formulation, 297-300
line spectral frequencies (LSF), 303-5
log-area ratios, 305-6
orthogonality principle, 291-92
prediction error, 301-3
reflection coefficients, 305
roots of the polynomial, 306
solution of the LPC equations, 292-300
spectral analysis via, 300-301
Linear pulse code modulation (PCM), 340-42
Linear time-invariant (LTI) systems, 207
eigensignals of, 209
with stochastic inputs, 267
Linear time-varying systems, 208
Line spectral frequencies (LSF), 303-5
Linguistic analysis, 692, 720-23
homograph disambiguation, 723, 724-25
noun phrase (NP) and clause detection, 723
POS tagging, 722-23
sentence tagging, 722
sentence type identification, 723
shallow parse, 723
Linguistic Data Consortium (LDC), 467
Linguistic forms, 859
Linguistics, co-references in, 882
Lips, 25
Liquid group, 42
Listening Effort Scale, 841
Listening Quality Scale, 841
LMS algorithm, 540
Load loss of signal level, 488
Localization issues, 696-97
Locus theory of speech production, 800
Log-area ratios, 305-6
Logical form, 67-68
Lognormal distribution, 97-98
Lombard effect, 480
Long-term prediction, 353
Look-ahead strategy, 661-62
Lossless compression, 338
Lossless tube concatenation, 284-88
Lossy compression, 338
Loudness, 740

Low-bit rate speech coders, 361-70
harmonic coding, 363-67
parameter estimation, 364-65
parameter quantization, 366-67
phase modeling, 365-66
mixed-excitation LPC vocoder, 362
waveform interpolation, 367-70
Lower bound of probability, 74
Low-frequency sounds, lateralization of, 31
Low-pass filter, bandwidth of, 240
Low-pass filters, digital, 229-30, 235
Low probability of intercept (LPI), 361
LPC, *See* Linear predictive coding (LPC)
LPC-cepstrum, 309-11
LPC vocoder, 353
LP-PSOLA, 832-33
LSE estimation, *See* Least squared error (LSE) estimation
LTI systems, *See* Linear time-invariant (LTI) systems
LTS conversion, 437, 728-30
LTS rules, 697
Lungs, 25

## M

McGurk effect, 69
Machine-learning methods, 56
McNemar's test, 148-49, 190
Magnitude-difference test, 119-20
Magnitude subtraction rule, 519
Mahalanobis distance, 166, 168
MAP, *See* Maximum a posteriori (MAP)
MAP estimation, 111-13
Marginal probability, 76, 77-78
Markov chain, 378-80
Masking, 30-31, 34-36, 349-50
Bark scale functions, 35
just noticeable distortion (JND), 35
spread-of-masking function, 35-36
temporal masking, 35-36
tone-masking noise, 35
Matched pairs test, 118-20, 148
Mathematical expressions, 718-19
MathML, 718
Maximal projection, 58
Maximum entropy models, 576-78
Maximum likelihood estimation (MLE), 73, 104-7, 134, 141, 168-69
Maximum likelihood estimator, 99
Maximum likelihood linear regression (MLLR), 447-50
vs. MAP, 450-51
Maximum mutual information estimation (MMIE), 134, 150-52, 156
defined, 151
Maximum phase signals, 309
Maximum a posteriori (MAP), 73, 111, 141, 331, 445-47, 854
Maximum substring matching problem, 420
MBROLA technique, 829
Mean, 79-81
Mean-ergodic process, 266
Mean opinion score (MOS), 338-39, 840
Mean squared error (MSE), 99
Mean vector, 84
Median, 81
Median smoother of order, 208
Mel-frequency cepstral coefficients (MFCC), 424-26
Mel frequency scale, 34
Message generation, 894-901
*See also* Response generation
Message generation box, 897
Metaunits, 658-59
Microphone, 936
Microphone arrays, 505-10
delay-and-sum beamformer, 505-6
goals of, 505
steering, 505
Microprosody, 767-68
Microsoft Dictation, 928-29
Microsoft Speech SDK 4.0, 937
Microsoft's speech API (SAPI), 921
Microsoft Whisper case study, 676-81
CFG search architecture, 676-77
*n*-gram search architecture, 677-81
Middle ear, 29
Mid-riser quantizer, 340
Mid-tread quantizer, 340
Minimum-classification-error (MCE), 156
Minimum cost-complexity pruning, 185-87
Minimum-error-rate decision rules, 135-38
Minimum-error-rate estimation, 134, 156-58
Minimum mean squared error (MMSE), 73, 99-104
for constant functions, 100
for linear functions, 101-2
for nonlinear functions, 102-4
Minimum mean square estimator, 99
Minimum phase signals, 309
Minimum squared error (MSE) estimation, 100
Minor phrase break, 751
MiPad case study, 945-52
evaluation, 949-51
iterations, 951-52
rapid prototyping, 948-49
specifying the application, 946-48
MITalk System, 735-36, 846
Mixed-excitation LPC vocoder, 362
Mixed excitation model, 289
Mixed initiative systems, 860
Mixture density estimation, 172
MMIE, *See* Maximum mutual information estimation (MMIE)
MMSE, *See* Minimum mean squared error (MMSE)
Mobile applications, 921
Mode, 81

Modified Discrete Cosine Transform (MDCT), 259
Modified Rhyme Test (MRT), 838
Modifiers, 61
Modular (component) testing, 731
Modulated Lapped Transform (MLT), 259
Money and currency, 716
Monolithic whole-system evaluation, 731
Morphological analysis, 693, 725-27
  algorithm, 727
  suffix and prefix stripping, 726-27
Morphological attributes, 55
Morphology, 56-57
  derivational, 56-57
  inflectional, 56
Move generator, 594
MP3, 371
MPEG, 351-52, 371
Multi-layer perceptrons, 160-63
Multimicrophone speech enhancement, 504-15
  blind source separation (BSS), 510-15
  microphone arrays, 505-10
    delay-and-sum beamformer, 505-6
    steering, 505
Multinomial distributions, 87-89
Multipass search, 663-74, 682
  *n*-best lists and word lattices, 664-66
  *n*-best search paradigm, 663
Multipass search vs. one-pass search, 673-74
Multiple tree combination, 730
Multiplexing delay, 339
Multirate signal processing, 248-51
  decimation, 248-49
  interpolation, 249-50
  resampling, 250-51
Multiresolution filterbanks, 254-254
Multistack search, 639
Multistyle training, 419
Multivariate distributions, 83-85
Multivariate Gaussian mixture density estimation, 172-75
Multivariate mixture Gaussian distributions, 93-95
Musical noise, 517
Musical pitch scales, and prosodic research, 32
MUSICAM, 352
Mutual information, 126-28

## N

Narrow-band filtering, 330
Narrow-band spectrograms, 282
Nasal, 42
Nasal cavity, 25
Nasal consonants, 43-44
Natural gradient, 513
Natural language, linguistic analysis of, 720-23
Natural language generation from abstract semantic input, 898
Natural language process (NLP) systems, 693-94
*N*-best lists, 664-66
*N*-best search paradigm, 663
Near-miss list, 158
Negative correlation, 83
Negotiation, 892
NETALK, 729
Neural networks, 134, 158, 457-59
  integrating with HMMs, 458-59
  recurrent, 457-58
  time delay neural network (TDNN), 458
Neural transduction process, 20
Neural units, 158
Neuromuscular signals, 20
Newton's algorithm, 155
*N*-gram language models, 558-60
*N*-gram pruning, 580-81
*N*-grams, search architecture, 677-81
*N*-gram smoothing, 562-74
  backoff smoothing, 565-70
    alternative backoff models, 568-70
    Good-Turing estimates and Katz smoothing, 565-67
  class *n*-grams, 570-74
    data-driven classes, 572-73
    rule-based classes, 571-72
  deleted interpolation smoothing, 564-65
  performance of, 573-74
Noise-canceling microphone, 490
Noiseless channels, 127
Noisy conditions, 330
Nonbranching hierarchies, 65
Noncausal Wiener filter, 522
Non-hierarchical relations, 65
Non-informative prior, 112
Nonlinear systems, 208
Nonstationary noise, modeling, 538-39
Normalized cross-correlation method, 327-29
Normalized LMS algorithm (NLMS), 501-2
Noun phrases (NPs), 58-59
Nouns, 54
*NP*-hard problem, 593
*N*-queens problem, 593, 598
Nucleus, 52
Number formats, 712-20
  account numbers, 716
  cardinal numbers, 717-18
  dates, 714-15
  money and currency, 716
  ordinal numbers, 717
  phone numbers, 712-14
  times, 715
Nyquist frequency, 243, 245

## O

Object-oriented programming, 869
Observable Markov model, 379-80
Obstruent, 43

Octaves, 32
Office applications, 921
Omnidirectional condenser microphones, 489-90
One-pass *n*-best and word-lattice algorithm, 669-70
One-pass vs. multipass search, 673-74
One-place predicates, 67
On-glides, 42
Onset, 52
Open-loop estimation, 356
Open POS categories, 54
Operations research problems, 604
Oral cavity, 25
Ordinal numbers, 717
Orthogonality principle, 291-92
Orthogonal processes, 263
Orthogonal variables, 83
Outer ear, 29
Out-Of-Vocabulary (OOV) word rate, 578
Outside probability, 556
Oval window, ear, 29
Overall quality tests, 840-41
  Absolute Category Rating (ACR), 841
  Listening Effort Scale, 841
  Listening Quality Scale, 841
  Mean Opinion Score (MOS), 840
Overlap-and-add (OLA) technique, 818-19
Overlapped evaluation scheme, 463
Oversampling, 246
Oversubtraction, 519

## P

Paired observations test, 114
Palatal consonants, 46
Palate, 46
Paradigmatic properties, 53
PARADISE framework, 903-6
Paragraphs, 702
Paralinguistic, use of term, 764
Parameter space, 98
Parametric Artificial Talker (PAT), 845
Parks McClellan algorithm, 236-38
Parsers, 721
Parse tree representations, 62-63
Parseval's theorem, 216
  for random processes, 268
Parsing algorithm, 545
Partial correlation coefficients (PARCOR), 299
Partition, 74
Part-whole, 66
Passive microphones, 496
Passive sentence, 62
Pattern recognition, 133-97
Pauses, 747-49
Pausing, 740
Penn Treebank project, 55
Perceived loudness, 30
Perceived pitch, 30
Perceptron training algorithm, 159
Perceptual attributes, sounds, 30
Perceptual Audio Coder (PAC), 351, 371
Perceptual linear prediction (PLP), 318-19
Perceptually-based distortion measures, 166
Perceptually motivated representations, 315-19
  bilinear transforms, 315-16
  mel-frequency cepstrum coefficients (MFCC), 316-18
  perceptual linear prediction (PLP), 318-19
Perceptual Speech Quality Measurement (PSQM), 844
Perceptual weighting, 357-58
Periodic lobe, 26
Periodic signals, 203
  cepstrum of, 311-12
Perplexity, 122, 560-62, 579
Personal Digital Assistants (PDAs), 930, 945
Phantom power, 488
Phantom trajectories, 463
Pharyngeal cavity, 25
Pharynx, 288
Phonemes, 20, 24, 36-38, 611
Phoneme trigram rescoring, 730
Phone numbers, 712-14
Phonetically balanced word list test, 839
Phonetic F0 (microprosody), 767-68
Phonetic languages, 692-93
Phonetic modeling, 428-39
  clustered acoustic-phonetic units, 432-36
  comparison of different units, 429-30
  context dependency, 430-31
  lexical baseforms, 436-39
Phonetics, 36-50
  allophones, 47-49
  clauses, 61-62
  coarticulation, 49-51
  consonants, 42-46
  lexical part-of-speech (POS), 53-56
  lexical semantics, 64-66
  logical form, 67-68
  morphology, 56-57
  parse tree representations, 62-63
  phonemes, 36-38
  phonetic typology, 46-47
  phrase schemata, 58-61
  semantic roles, 63-64
  semantics, 58
  sentences, 61-62
  speech rate, 49-51
  syllables, 51-52
  syntactic constituents, 58
  syntax, defined, 58
  vowels, 39-42
  word classes, 57
  words, 53-57
Phonetic typology, 46-47
Phonological phrases, 749

Phonology, 36-50
Phrase schemata, 58-61
Phrase-structure diagram, 63
Physical vs. perceptual attributes of sounds, 30-32
Physiology of the ear, 29-32
Pickup pattern, microphone, 489
Piezoelectric microphones, 497
Piezoresistive microphones, 497
Pink noise, 270, 478
Pitch, 25, 30, 47, 740
  autocorrelation method, 324-27
  normalized cross-correlation method, 327-29
  role of, 324-32
  signal conditioning, 329-30
  tracking, 330-32
Pitch generation, 763-82
  accent termination, 770
  attributes of pitch contours, 764-68
  baseline F0 contour generation, 768-69
  corpus-based F0 generation, 779-82
    F0 contours indexed by parsed text, 779-81
    F0 contours indexed by ToBI, 781-82
    transplanted prosody, 779
  declination, 766-67
  evaluations/improvements, 773-74
  F0 contour interpolation, 772-73
  gradient prominence, 765-66
  interface to synthesis module, 773
  parametric F0 generation, 774-75
  phonetic F0 (microprosody), 767-68
  pitch range, 764-65, 770-71
  prominence determination, 771-72
  superposition models, 775-76
  ToBI realization models, 777-78
  tone determination, 770
Pitch prediction, 356-57
Pitch range, 764-65, 770-71
Pitch-scale modification epoch calculation, 825
Pitch-scale time-scale epoch calculation, 827
Pitch synchronous analysis, 283, 302-3
Pitch synchronous overlap and add (PSOLA), 820-23, 831, 847
  problems with, 829-31
    amplitude mismatch, 830
    buzzy voiced fricatives, 830
    phase mismatches, 829
    pitch mismatches, 830-31
  spectral behavior of, 822-23
Pitch tracking, 330-32
Pitch tracking errors, 828
Plan-based dialog modeling, 914
Plan-based systems, 888-92
Plan libraries, 889
Plosive, 42
Plosive consonant, 42-43
Poisson distributions, 89
Poles, 213
Pole-zero filters, cepstrum of, 308-98
Polymorphic linguistic context assignment, 656-57
Polysemy, 65
Positive correlation, 83
Positive-definite function, 262
POS tagging, 56, 722-23
Posterior probability, 135, 142, 156
Postfiltering, 357-58
Post-lexical rules, 735
Postmodifiers, 58-61
Power function, 114
Power spectral subtraction rule, 519
Power spectrum, 216
Predicate, 61, 67
Predicate logic, 68
Pre-emphasis filtering, 235, 320
Preference tests, 842
Prefix nodes, 658
Prefix trees, 647
Premodifiers, 58-59
Prepositions, 54, 60
Pressure gradient microphones, 496
Pressure microphones, 496
Prime-factor algorithm, 224
Principal-component analysis (PCA), 426
Priority entity memory, 882-83
Prior probability, 133, 135, 140
Probabilistic CFGs (PCFGs), 554
Probabilistic context-free grammars, 554-58
Probability density function (pdf), 78, 261
Probability function (pf), 77
Probability mass function (pmf), 77
Probability theory, 73-131
  Bayes' rule, 75-78
  binomial distributions, 86
  chain rule, 75, 77-78
  conditional probability, 75-76
  correlation, 82-83
  covariance, 82-83
  gamma distributions, 90-91, 95
  Gaussian distributions, 92-98
  geometric distributions, 86-87
  law of large numbers, 82
  marginal probability, 76, 77-78
  mean, 79-81
  multinomial distributions, 87-89
  multivariate distributions, 83-85
  Poisson distributions, 89
  probability density function (pdf), 78
  random variables, 77-79
  random vectors, 83-85
  uniform distributions, 85
  variance, 79-81
Prominence determination, 771-72
Prompting strategy, 943
Pronouns, 54
Pronunciation trees, 648-50

Proper noun, 53-54
Property inheritance, 871
Propositional phrases (PPs), 59-60
Prosodic analysis module, 7
Prosodic modification of speech, 818-31
  epoch detection, 828-29
  evaluation of TTS systems, 834-44
    automated tests, 843-44
    Diagnostic Rhyme Test (DRT), 837
    functional tests, 842-43
    glass-box vs. black-box evaluation, 835-36
    global vs. analytic assessment, 836
    Haskins Syntactic Sentence Test, 839
    human vs. automated, 835
    intelligibility tests, 837-39
    judgment vs. functional testing, 835-36
    laboratory vs. field, 835
    Modified Rhyme Test (MRT), 838
    overall quality tests, 840-41
    phonetically balanced word list test, 839
    preference tests, 842
    Semantically Unpredictable Sentence Test, 837
    symbolic vs. acoustic level, 835
  pitch-scale modification epoch calculation, 825
  pitch-scale time-scale epoch calculation, 827
  pitch synchronous overlap and add (PSOLA), 820-23, 831, 847
    problems with, 829-31
    spectral behavior of, 822-23
  source-filter models for prosody modification, 831-34
    LP-PSOLA, 832-33
    mixed excitation models, 832-34
    prosody modification of the LPC residual, 832
    voice effects, 834
  synchronous overlap and add (SOLA), 818-19
  synthesis epoch calculation, 823-24
  time-scale modification epoch calculation, 826-27
  waveform mapping, 827-28
Prosodic phrases, 749-51
Prosodic transcription systems, 759-61
Prosody, 739-91, 943
  and character, 744
  duration assignment, 761-63
    CART-based durations, 763
    rule-based methods, 762-63
  generation, 721-22
  generation schematic, 743-44
  loudness, 740
  pausing, 740
  pitch, 740
  pitch generation, 763-82
    accent termination, 770
    attributes the pitch contours, 764-68
    baseline F0 contour generation, 768-69
    corpus-based F0 generation, 779-82
    declination, 766-67
    evaluations/improvements, 773-74
    F0 contour interpolation, 772-73
    gradient prominence, 765-66
    interface to synthesis module, 773
    parametric F0 generation, 774-75
    phonetic F0 (microprosody), 767-68
    pitch range, 764-65, 770-71
    prominence determination, 771-72
    superposition models, 775-76
    ToBI realization models, 777-78
    tone determination, 770
  prosody markup languages, 783-85
  rate/relative duration, 740
  role of understanding, 740-44
  speaking style, 744-45
    character, 744
    emotion, 744-45
  symbolic, 745-61
    accent, 751-53
    pauses, 747-49
    prosodic phrases, 749-51
    prosodic transcription systems, 759-61
    tone, 753-57
    tune, 757-59
Prosody markup languages, 783-85
PROSPA, 759
Pruning, 609
Pruning error, 675
PSOLA, *See* Pitch synchronous overlap and add (PSOLA)
Psychoacoustics, 30
Pulse code modulation (PCM), 271, 340-42
Pure tones, 31
Push-down automation, 548
Push-to-talk model, 422-23
P-value, 115-16

## Q

Quantization noise, 246
Questioned noun phrase, 61

## R

Radix-2 FFT, 222, 223
Randomness, 72
Random noise, 276
Random variables, 77-79
  expectation of, 79
Random vectors, 83-85
Rapidly evolving waveforms (REW), 368-70
Rapid prototyping, 948-49
Rate/relative duration, 740
Read speech acoustic models, 857
Real cepstrum, 307-8
Real-time cepstral normalization, 525
Recognition problem, 554
Rectangular window, 230-31
Recurrent neural networks, 457-58

Recursive least squares (RLS) algorithm, 504
Recursive transition network (RTN), 548, 613-14
Reflection coefficients, 296, 299, 305
Region of convergence (ROC), 211-12
Regular Pulse Excited-Linear Predictive Coder (RPE-LPC), 360
Relative clauses, 61
Relative expressions, 871
Relative frequency, 74
Relative spectral processing (RASTA), 525
Renditions, 899-901
Repetitive stress injury (RSI), 929
Residual signal, 301
Resonances of vocal tract, excitation of, 27
Response generation, 894-901
  message generation box, 897
  natural language generation from abstract semantic input, 898
  response content generation, 895-99
  template systems for, 897
Retraining on compensated features, 537-38
Retraining on corrupted speech, 528-39
Retroflex liquid, 42
Reverberation, 480-82
Ribbon microphones, 497
Rich-Get-Richer (RGR) strategy, 662
Right-sized tree, 184-89
  cross-validation, 188-89
  independent test sample estimation, 187-88
  minimum cost-complexity pruning, 185-87
RLS algorithm, 503-4
Robust parsing, 874-78
Roll-off, 281
Rule-based duration-modeling methods, 56
Rule-based speech synthesis systems, 795-96
  CPU resources, 795
  delay, 795
  memory resources, 795
  pitch control, 795
  variable speed, 795
  voice characteristics, 796

## S

Sadness, and speech, 745
Sample mean, 82
Sample variance, 82
Sampling theorem, 243-45
SAM system, 913-14
Scalable coders, 371
Scalar frequency domain coders, 348-52
  consumer audio, 351-52
  Digital Audio Broadcasting (DAB), 352
  frequency domain, advantages of, 348-49
  masking, 349-50
  transform coders, 350-51
Scalar waveform coders, 340-48
  adaptive PCM, 344-45
  differential quantization, 345-48
  linear pulse code modulation (PCM), 340-42
  μ-law and A-law PCM, 342-44, 348
Screen reader, 929
Search, defined, 592
Search-algorithm evaluation, 674-76
Search algorithms:
  beam, 606-8
  best-first, 602-6
  blind graph, 597-601
  breadth-first, 600-601
  depth-first, 598-99
  forward-backward, 670-73
  large vocabulary, 645-85
  speech-recognition, 608-12
  combining acoustic and language models, 610
  continuous speech recognition, 611-12
  decoder basics, 609
  isolated word recognition, 610-11
  tree-trellis forward-backward, 671
Search error, 675
Second-order IIR filters, 241-42
Second-order resonators, 242
Segment models, 459-60
Segment-model weight, 462
Selectivity, of grammar, 546
Semantically Unpredictable Sentence Test, 837
Semantic authoring, 862
Semantic classes, 948-49
Semantic grammars, 585
Semantic language model, 879
Semantic parser, 854-55
Semantic representation, 867-73
  conceptual graphs, 872-73
  functionality encapsulation, 871-72
  property inheritance, 871
  semantic frames, 867-69
  type abstraction, 869-71
Semantic roles, 63-64
Semantics:
  defined, 58
  language, 545
  lexical, 64-66
Semicontinuous HMMs, 396-98
Semi-tones, 32
Semivowels, 42
Senones, 433-36, 467, 809
Senone sequence, 658
Sentence interpolation, 873-80
  robust parsing, 874-78
  defined, 875
  statistical pattern matching, 878-80
  syntactic grammars, 877
Sentence interpretation, 7
Sentence interpretation module, 855
Sentence-level stress, 431

Sentences, 61-62, 702-4
diagramming in parse trees, 63
Sentence tagging, 722
Sentence type identification, 723
Shades of meaning, 67
Shallow parse, 723
Shannon's channel coding theorem, 127
Shannon's source coding theorem, 124-25, 128
Sharing, 609
Sharing tails, 655-56
Shimmer, 768
Short-term prediction, 353
Short-time Fourier analysis, 276-83
pitch-synchronous analysis, 283
spectrograms, 281-83
Sigma-delta A/D, 246
Sigma-delta modulation, 346
Signal acquisition, 422
Signal conditioning, 329-30
Signal processing module, 421-28
end-point detection, 422-24
feature transformation, 426-28
mel-frequency cepstral coefficients (MFCC), 424-26
signal acquisition, 422
Signals, 201
Signal-to-noise ratio (SNR), 339, 486, 489
Significance testing, 98, 113-20
goodness-of-fit test, 116-18
level of significance, 114-15
magnitude-difference test, 119-20
matched pairs test, 118-20
normal test, 115-16
sign test, 119
Z test, 115-16
Sign test, 119
Silences between words, handling, 621-22
Similars, 65-66
Simple questions, 177
Sinc function, 229-30
Single-layer perceptrons, 159-60
Singleton questions, 177
Single-word subpath, 655
Sinusoidal coding, 371
Sinusoidal systems, 203-5
Slope overload distortion, 346
Slot inheritance, 885
Slowly evolving waveform (SEW), 367-70
SLU, *See* Spoken language understanding (SLU) systems
Smart phones, 930
SNR, *See* Signal-to-noise ratio (SNR)
Soft palate, 25
Sound, 21-23
Sound Blaster, 936
Sound pressure level (SPL), 23
Source coding theorem, 124-26
Source-filter models for prosody modification, 831-34
Source-filter models of speech production, 288-90
Source-filter separation, via the cepstrum, 314-15
Speak & Spell, 271
Speaker-adaptive training techniques, 419
Speaker-dependent speech recognition, 418-19
Speaker-independent speech recognition, 418
Speaker recognition, 931
Speaker variability, 418-19
Speaking style, 744-45
character, 744
emotion, 744-45
Speaking turn, 861
Specificity ordering conflict resolution strategy, 182
Specifier position, 61
Spectral analysis via linear predictive coding (LPC), 300-301
Spectral leakage, 279
Spectral subtraction, 516-19
Spectrograms, 27-28, 276, 281-83
Speech:
defined, 283
interfacing with computers, 1
prosodic modification of, 818-31
supplemented by information streams, 2
using as an add-on feature, 941
Speech acts, 705-6
Speech-act theory, 914
Speech coding, 337-74
code excited linear prediction (CELP), 353-61
adaptive codebook, 356-57
analysis by synthesis, 353-56
LPC vocoder, 353
parameter quantization, 358-59
perceptual weighting/postfiltering, 357-58
pitch prediction, 356-57
standards, 359-61
coder delay, 339
low-bit rate speech coders, 361-70
harmonic coding, 363-67
mixed-excitation LPC vocoder, 362
waveform interpolation, 367-70
scalar frequency domain coders, 348-52
consumer audio, 351-52
Digital Audio Broadcasting (DAB), 352
masking, 349-50
transform coders, 350-51
scalar waveform coders, 340-48
adaptive PCM, 344-45
differential quantization, 345-48
linear pulse code modulation (PCM), 340-42
μ-law and A-law PCM, 342-44, 348
speech coder attributes, 338-39
Speech communication, history of, 1
Speech end-point detector, 423
Speech interaction, modes of, 933-34

Speech interface design, 931-43
general principles of, 931-37
human limitations, 932
modes of interaction, 933-34
technological considerations, 935-36
user accommodation, 933
handling errors, 937-41
error detection and correction, 938-39
feedback and confirmation, 939-41
internationalization, 943-45
Speech inversion problem, 803-4
Speech perception, 29-36
Speech processing:
digital signal processing, 201-73
speech coding, 337-74
speech signal representations, 275-336
Speech production, 24-28
acoustical model of, 283-90
articulators, 24-25
formants, 27-28
frequency analysis, 32-34
masking, 34-36
physical vs. perceptual attributes of sounds, 30-32
physiology of the ear, 29-32
spectrograms, 27-28
speech perception, 29-36
voicing mechanism, 25-27
*See also* Acoustical model of speech production
Speech production process, start of, 19
Speech rate, 49-51
Speech recognition, 2, 3, 375-685, 862
acoustic modeling, 415-75
context variability, 417
environment variability, 419
scoring acoustic features, 439-43
speaker variability, 418-19
speech recognition errors, 419-21
style variability, 418
variability in speech signals, 416-19
hidden Markov models (HMM), 377-413, 416
Baum-Welch algorithm, 389-93
continuous mixture density, 394-96
decoding, 387-89
definition of, 380-93
deleted interpolation, 401-3
dynamic programming, 384-85
dynamic time warping (DTW), 383-85
estimating parameters, 389-93
evaluating, 385-87
forward algorithm, 385-87
forward-backward algorithm, 389-93
initial estimates, 398-99
limitations of, 405-9
Markov chain, 378-80
model topology, 399-401
observable Markov model, 379-80
parameter smoothing, 403-4
practical issues concerning, 398-405
probability representations, 404-5
semicontinuous, 396-98
training criteria, 401
Viterbi algorithm, 387-89
phonetic modeling, 428-39
clustered acoustic-phonetic units, 432-36
comparison of different units, 429-30
context dependency, 430-31
lexical baseforms, 436-39
signal processing module, 421-28
end-point detection, 422-24
feature transformation, 426-28
mel-frequency cepstral coefficients (MFCC), 424-26
signal acquisition, 422
speech recognition errors, 419-21
word error rate, 420
word recognition errors, types of, 420
Speech recognition search algorithms, 608-12
combining acoustic and language models, 610
continuous speech recognition, 611-12
decoder basics, 609
isolated word recognition, 610-11
Speech recognition system, 4-5
basic system architecture of, 5
components of, 4
source-channel model for, 5
vocabulary, 58
Speech signal representations, 275-336
acoustical model of speech production, 283-90
glottal excitation, 284
lossless tube concatenation, 284-88
mixed excitation model, 289
source-filter models of speech production, 288-90
cepstrum, 306-15
cepstrum vector, 309
complex, 307-8
LPC-cepstrum, 309-11
of periodic signals, 311-12
of pole-zero filters, 308-98
real, 307-8
source-filter separation via, 314-15
of speech signals, 312-13
formant frequencies, 319-23
statistical formant tracking, 320-23
linear predictive coding (LPC), 290-306
autocorrelation method, 295-96
covariance method, 293-94
equivalent representations, 303-6
lattice formulation, 297-300
line spectral frequencies (LSF), 303-5
log-area ratios, 305-6
orthogonality principle, 291-92
prediction error, 301-3
reflection coefficients, 305
roots of the polynomial, 306

Speech signal representations, linear predictive coding *(cont.)*
solution of the LPC equations, 292-300
spectral analysis via, 300-301
perceptually motivated representations, 315-19
bilinear transforms, 315-16
mel-frequency cepstrum coefficients (MFCC), 316-18
perceptual linear prediction (PLP), 318-19
pitch:
autocorrelation method, 324-27
normalized cross-correlation method, 327-29
pitch tracking, 330-32
role of, 324-32
signal conditioning, 329-30
short-time Fourier analysis, 276-83
pitch-synchronous analysis, 283
spectrograms, 281-83
Speech signals, 5, 20
cepstrum of, 312-13
context variability, 417
environment variability, 419
speaker variability, 418-19
style variability, 418
variability in, 416-19
Speech synthesis, 6, 793-852
articulatory speech synthesis, 793, 803-4
attributes of, 794-96
concatenative synthesis with no waveform modification, 794
concatenative synthesis with waveform modification, 795
limited-domain waveform concatenation, 794
rule-based systems, 795-96
concatenative speech synthesis, 793-94
choice of unit, 805-8
context-dependent phonemes, 808-9
context-independent phonemes, 806-7
diphones, 807-8
optimal unit string, 810-17
subphonetic units (senones), 809
syllables, 809
unit inventory design, 817-18
word and phrase, 809
data-driven synthesis, 794, 803
formant speech synthesis, 793, 796-804
cascade model, 797
formant generation by rule, 800-803
Klatt's cascade/parallel formant synthesizer, 797-802
locus theory of speech production, 800
parallel model, 797
waveform generation from formant values, 797-99
prosodic modification of speech, 818-31
epoch detection, 828-29
pitch-scale modification epoch calculation, 825
pitch-scale time-scale epoch calculation, 827
pitch synchronous overlap and add (PSOLA), 820-23, 847
synchronous overlap and add (SOLA), 818-819
synthesis epoch calculation, 823-24
time-scale modification epoch calculation, 826-27
waveform mapping, 827-28
synthesis by rule, 794
Speech-to-speech translation, 3
Split-radix algorithm, 223
Splits, 182
Spoken language, 19
Spoken language interface, 2-3
Spoken language processing, 4, 133
Spoken language structure, 19-72
Spoken language system, 2
Spoken language system architecture, 4-8
automatic speech recognition, 4-6
spoken language understanding, 7-8
text-to-speech conversion, 6-7
Spoken language understanding, 7-8
basic system architecture of, 8
Spoken language understanding (SLU) systems, 853-918, 945
assumptions, 854
content, 854
context, 854
dialog management, 886-94
dialog grammars, 887-88
plan-based systems, 888-92
dialog structure, 859-67
attentional state, 859
dialog (speech) acts, 861-66
intentional state, 859
linguistic forms, 859
task knowledge, 859
units of dialog, 860-61
world knowledge, 859
dialog system, 854-55
dialog manager, 855
discourse analysis, 855
semantic parser, 854-55
discourse analysis, 881-86
resolution by NLP, 883-85
resolution of relative expression, 882-85
Dr. Who case study, 906-13
evaluation, 901-6
in the ATIS task, 901-3
PARADISE framework, 903-6
historical perspective, 913-14
intent, 854
rendition, 899-901
response generation, 894-901
concept-to-speech rendition, 899-901
natural language generation from abstract semantic input, 898
response content generation, 895-99
semantic representation, 867-73

conceptual graphs, 872-73
functionality encapsulation, 871-72
property inheritance, 871
semantic frames, 867-69
type abstraction, 869-71
sentence interpolation, 873-80
robust parsing, 874-78
statistical pattern matching, 878-80
syntactic grammars, surface linguistic variations in, 877
written vs. spoken languages, 855-58
communicative prosody, 858
disfluency, 857
style, 856-57
Spoken menus, 942
Spread-of-masking function, 35-36
Spread spectrum, 360-61
Stable LTI system, 211
Stack decoding, 592
advantage of, 628
defined, 627
formulating in a tree search framework, 629
Stack decoding (A* search), 626-39
admissible heuristics for remaining path, 630-31
extending new words, 631-34
fast match, 634-38
multistack search, 639
stack pruning, 638-39
Stack pruning, 638-39
Standard deviation, 80
Standard Gaussian distributions, 92-93
State-space search paradigm, 592
Stationary processes, 264-67
ergodic processes, 265-67
Stationary signal, 276
Statistical formant tracking, 320-23
Statistical inference, 98, 113
Statistical language models, 583
Statistical pattern matching, 878-80
Statistical pattern recognition, 190
Statistics, 73-131
Stochastic language models, 546, 554-60
*n*-gram language models, 558-60
probabilistic context-free grammars, 554-58
Stochastic processes, 260-70
continuous-time, 260
discrete-time, 260
LTI systems with stochastic inputs, 267
noise, 269-70
power spectral density, 268-69
stationary processes, 264-67
statistics of, 261-64
Stop, 43
Stress, 751
Stressed vowels, 430-31
Strict-sense stationary (SSS), 264
Style, 856-57
Style variability, 418
Subgoals, 894
Sub-harmonic errors, 330
Subject, sentence, 61
Subphonetic units (senones), 809
Subscripting, 67
Substitution errors, 420
Subtree dominance, 656
Subtree isomorphism, 654
Subtree polymorphism, exploiting, 656-58
Successor operator, 594
Sum-of-squared-error (SSE), 99, 160
Superposition models, 775-76
Supervised learning, 134, 141
Surrogate questions, 182
SWITCHBOARD Shallow-Discourse-Function Annotation SWBD-DAMSL, 865-66
Syllable parse tree, 52
Syllables, 20, 51-52, 430, 809
Syllables centers, 52
Symbolic prosody, 745-61
accent, 751-53
pauses, 747-49
prosodic phrases, 749-51
prosodic transcription systems, 759-61
tone, 753-57
tune, 757-59
Symmetrical loss function, 136
Symmetric channel, 127
Synchronous overlap and add (SOLA), 818-19
Syntactic constituents, 58
Syntactic theory, 69
Syntagmatic properties, 53
Syntax:
defined, 58
language, 545
Synthesis-by-rule, 794, 796
Synthesis epoch calculation, 823-24
System initiative, 860


## T


Tag question, 62
Tags, 7
Tail area, 115
Tap and Talk interface, 934, 947-48, 951
Task knowledge, 859
TDMA Interim Standard 54, 360
Telecommunication Industry Association (TIA), 360
Telephone speech, 338
Telephony applications, 924-26
Temporal masking, 35-36
Testing set, 141
Test procedure, 114
Text analysis phase, 7
Text normalization (TN), 692, 706-20
abbreviations, 709-12
acronyms, 711-12

Text normalization (TN) *(cont.)*
domain-specific tags, 718-20
chemical formulae, 719
mathematical expressions, 718-19
miscellaneous formats, 719-20
evaluation, 730-32
Festival case study, 732-35
historical perspective, 735-36
letter-to-sound (LTS) conversion, 728-30
linguistic analysis, 720-23
and closed-class function words, 722
homograph disambiguation, 723, 724-25
noun phrase (NP) and clause detection, 723
POS tagging, 722-23
sentence tagging, 722
sentence type identification, 723
shallow parse, 723
morphological analysis, 725-27
algorithm, 727
suffix and prefix stripping, 726-27
number formats, 712-20
account numbers, 716
cardinal numbers, 717-18
dates, 714-15
money and currency, 716
ordinal numbers, 717
phone numbers, 712-14
times, 715
Text and phonetic analysis, 689-738
American-English vocabulary relevant to, 698
data flow, 694-97
skeleton, 695
defined, 692
document structure detection, 692, 699-706
grapheme-to-phoneme conversion, 692-93
homograph disambiguation, 693
letter-to-sound (LTS) conversion, 693
lexicon, 697-98
linguistic analysis, 692
localization issues, 696-97
modules, 692-94
morphological analysis, 693
natural language process (NLP) systems, 693-94
phonetic languages, 692-93
text normalization (TN), 692, 706-20
Text-to-speech (TTS) conversion, 6-7
Text-to-speech (TTS) system, 687-850
basic system architecture of, 6
goals of, 689-90
phonetic analysis component, 7
prosody, 739-91
speech synthesis, 793-850
speech synthesis component, 7
tags, 7
text analysis component, 6-7
text and phonetic analysis, 689-738
Text-to-speech (TTS) system evaluation, 834-44
automated tests, 843-44
Diagnostic Rhyme Test (DRT), 837
functional tests, 842-43
glass-box vs. black-box evaluation, 835-36
global vs. analytic assessment, 836
Haskins Syntactic Sentence Test, 839
historical perspective, 844-47
human vs. automated, 835
intelligibility tests, 837-39
judgment vs. functional testing, 835-36
laboratory vs. field, 835
Modified Rhyme Test (MRT), 838
overall quality tests, 840-41
Absolute Category Rating (ACR), 841
Listening Effort Scale, 841
Listening Quality Scale, 841
Mean Opinion Score (MOS), 840
phonetically balanced word list test, 839
preference tests, 842
Semantically Unpredictable Sentence Test, 837
symbolic vs. acoustic level, 835
TFIDF information retrieval measure, 576
Third generation (3G) systems, 361
Threshold of hearing (TOH), 22
Threshold value, likelihood ratio, 139
Throat, 25
TIA/EIA/IS54, 360
TIA/EIA/IS-127-2, 361
TIA/EIA/IS-733-1, 361
TILT, 760
Timbre, 25, 32
Time delay neural network (TDNN), 458
Time Division Multiple Access (TDMA), 360
Time-scale modification epoch calculation, 826-27
Time-synchronous Viterbi beam search, 622-26
algorithm, 627
use of beam, 624-25
Time-synchronous Viterbi search, 666-67
TN, *See* Text normalization (TN)
ToBI realization models, 777-78
ToBI (Tones and Break Indices) system, 749-50, 754, 777
boundary tolerance, 756
intermediate phrasal tones, 756
pitch accent tones, 755
Toeplitz matrix, 296
Toll quality, 344
Tone, 753-57
Tone determination, 770
Tone-masking noise, 35
Tongue, 25
Top-down chart parsing, 549-51
top-down vs., 549-50
Topic-adaptive models, 575-76
Trachea, 25
Trainability, 145

Training corpus, 559
Training problem, 554
Training set, 141, 419-20
Transducers, acoustical, 486-97
Transfer function, 210
Transformation models, 454-55
Transform coders, 350-51, 371
Transform-domain LMS algorithm, 502-3
Transition network, 548
Transmission delay, 339
Transparent quality, 358
Tree-banked data, 878
Tree lexicon, efficient manipulation of, 646-59
Tree structure, 646
Tree-trellis forward-backward search algorithms, 671
Triangular windows, 280
Trigram grammar, 465
Trigrams, 559, 583
  search space with, 619-20
Trilled *r* sound, 47
Triphone model, 430
Triphones, 808
TTS models, 949
TTS system, *See* Text-to-speech (TTS) system
Tune, 757-59
Turing machine, 548
Turing test, 3, 843
Turn memories, 882
Twiddle factors, 224
Two-band conjugate quadrature filters, 251-54
Twoing rule, 181
Two-place predicates, 67
Two-tailed test, 115-16
Type abstraction, 869-71
Type-I filter, 233
Type-n-Talk system, 847

## U

U method, 147
Uncertainty, 73, 121
Uncorrelated orthogonal processes, 263
Undergeneration, 876
Understandability, of grammar, 546
Unicode, 36
Unidirectional microphones, 494-96
Unification grammar, 584
Unified frame- and segment-based models, 462-64
Uniform distributions, 85
Uniform prior, 112
Uniform quantization, 340
Uniform search, 597
Unigram, 559
  search space with, 616-17
Unimodal distribution, 95
Uniquely decipherable coding, 125
United States Public Switched Telephone Network (PSTN), 371
Units, choice of, in concatenative speech synthesis, 805-8
Units of dialog, 860-61
Universal encoding scheme, 126
Universal Mobile Telecommunications System (UMTS), 361
Unknown word, defined, 563
Unstressed vowels, 430
Unsupervised estimation methods, 163-75
  EM algorithm, 170-72
  multivariate Gaussian mixture density estimation, 172-75
  vector quantization (VQ), 164-70
Unsupervised learning, 141
Upper bound of probability, 74
U.S. Defense Advanced Research Projects Agency (DARPA), 467
User expectations, managing, 942
User initiative, 860, 867
Utterance unit, 861

## V

Variance, 79-81
Vector quantization (VQ), 164-70, 191
  distortion measures, 164-66
  EM algorithm, 170-72
  *K*-means algorithm, 166-69
  LBG algorithm, 169-70
Vector Taylor series, 535-37
Velar consonants, 46
Velum, 25
Verbs, 54
Verbs phrases (VPs), 59-61
V-fold cross-delegation, 188-89
V-fold cross validation, 147
ViaVoice (IBM), 926
Viterbi algorithm, 387-89, 409, 609
Viterbi approximation, 623
Viterbi beam search, 625-26
Viterbi decoder, 592
Viterbi forced alignment, 630
Viterbi stack decoder, 592
Viterbi trellis, 624
Vocabulary independence, 433
Vocabulary selection, 578-80
Vocal cords, 25
Vocal fold cycling at the larnyx, 26
Vocal fry, 330
Vocal tract normalization (VTN), 427
Voder, 6
Voice conversion, 834
Voice effects, 834
Voice FONCARD (Sprint), 931
Voiceless plosive consonants, 43
Voice over Internet protocol (Voice over IP), 359
Voice portals, 925
VoiceXML, 921

Voice Xpress, 926
Voicing mechanism, 25-27
Vowels, 24, 39-42
  Japanese, 46-47
VQ, *See* Vector quantization (VQ)

## W

Wall Street Journal (WSJ) Dictation Task, 11-12
Waveform-approximating coders, 361
Waveform interpolation, 367-70, 371
Waveform mapping, 827-28
Waveforms, fundamental frequency, 27
Web pages, 705
Whisper case study, 464-65
Whispering effect, 834
White noise, 269-70, 478-79
Whole-word models, difficulty in building, 428
Wh-question, 62
Wide-band spectrograms, 282
Wideband speech, 338
Wide-sense stationary (WSS), 265
Wiener filtering, 520-22, 540
  noncausal, 522
Wiener-Hopf equation, 521
Wiener-Khinchin theorem, 269
Window design filter, 235-36
Window design FIR lowpass filters, 235-36
Window function, 255, 277-78
Window functions, 230-32
  generalized Hamming window, 231-32
  rectangular window, 230-31
Wizard-of-Oz (WOZ) experimentation, 950
Word classes, 57
Word-dependent *n*-best and word-lattice algorithm, 667-70
Word error rate, 420-21
  algorithm to measure, 421
Word error rate comparisons, humans vs. machines, 12
Word-final unit, 659
Word graphs, 664-66
Word-initial unit, 658
Word-lattice algorithm:
  one-pass *n*-best and, 669-70
  word-dependent *n*-best and, 667-70
Word-lattice generation, 672-73
Word lattices, 664-66
Word-level stress, 431
Word recognition errors, types of, 420
Words, 20, 53-57
  natural affinities/disaffinities, 65
Word-spotting applications, 454
World knowledge, 859
Written vs. spoken languages, 855-58
  disfluency, 857
  style, 856-57

## X

$\chi^2$ distributions, 95-96
*X*-bar theory, 885
XML, 699-700
X-template, 58

## Y

Yes-no question, 62
Yule-Walker equations, 291-92, 299

## Z

Zero-mean process, 262
Zero-one loss function, 136
Zero padding, 227, 280
Zeros, 213
*Z* test, 115-16
*Z*-transforms, 211-12
  of elementary functions, 212-15
    inverse *z*-transform of rational functions, 213-15
    left-sided complex exponentials, 213
    right-sided complex exponentials, 212-13
  properties of, 215-17
    convolution property, 215
    power spectrum and Parseval's theorem, 216

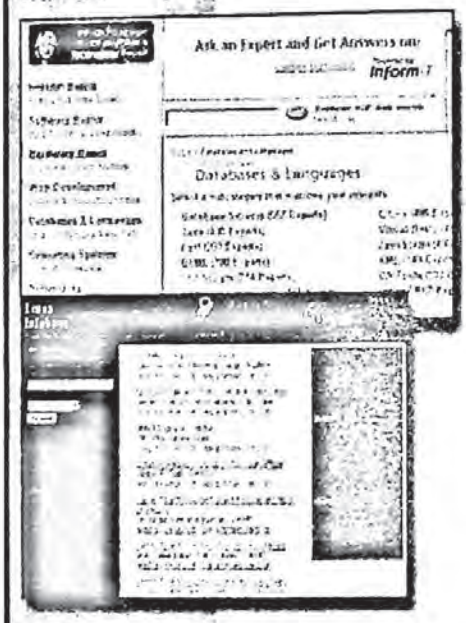
Inform

Solutions from experts you know and trust.

Articles | Free Library | eBooks | Expert Q & A | Training | Career Center | Downloads | MyInformIT

Login | Register | About InformIT

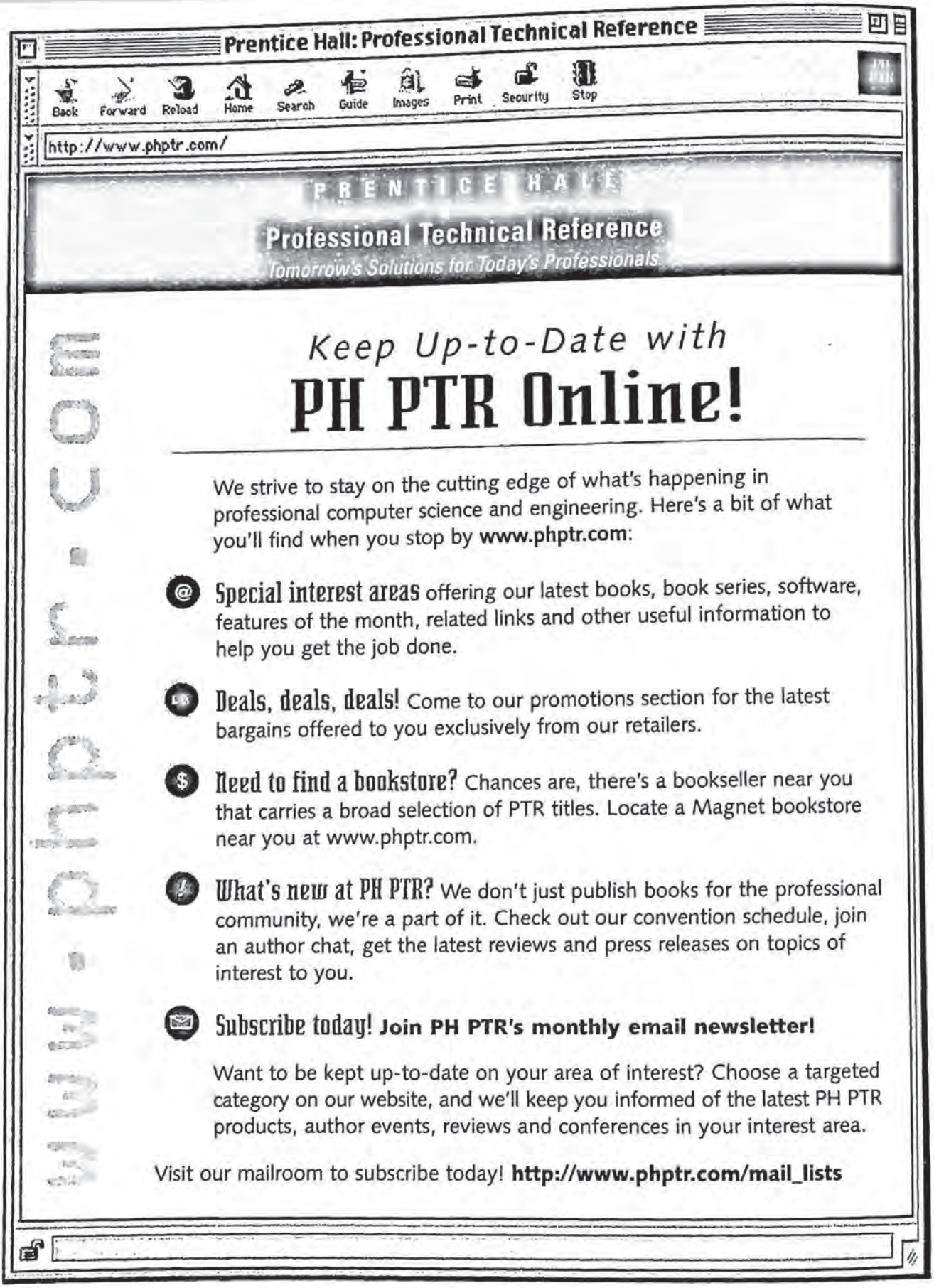
www.informit.com



- Free, in-depth articles and supplements
- Master the skills you need, when you need them
- Choose from industry leading books, ebooks, and training products
- Get answers when you need them - from live experts or InformIT's comprehensive library
- Achieve industry certification and advance your career

Visit InformIT today and get great content from PH PTR

Prentice Hall and InformIT are trademarks of Pearson plc /
Copyright © 2000 Pearson


Prentice Hall: Professional Technical Reference
Back
Forward
Reload
Home
Search
Guide
Images
Print
Security
Stop
http://www.phptr.com/
PRENTICE HALL
Professional Technical Reference
Tomorrow's Solutions for Today's Professionals
http://www.phptr.com
Keep Up-to-Date with
PH PTR Online!
We strive to stay on the cutting edge of what's happening in professional computer science and engineering. Here's a bit of what you'll find when you stop by www.phptr.com:
Special interest areas offering our latest books, book series, software, features of the month, related links and other useful information to help you get the job done.
Deals, deals, deals! Come to our promotions section for the latest bargains offered to you exclusively from our retailers.
Need to find a bookstore? Chances are, there's a bookseller near you that carries a broad selection of PTR titles. Locate a Magnet bookstore near you at www.phptr.com.
What's new at PH PTR? We don't just publish books for the professional community, we're a part of it. Check out our convention schedule, join an author chat, get the latest reviews and press releases on topics of interest to you.
Subscribe today! Join PH PTR's monthly email newsletter!
Want to be kept up-to-date on your area of interest? Choose a targeted category on our website, and we'll keep you informed of the latest PH PTR products, author events, reviews and conferences in your interest area.
Visit our mailroom to subscribe today! http://www.phptr.com/mail_lists

Voice/Speech Recognition Technology

XUEDONG HUANG

ALEX ACERO

HSIAO-WUEN HON

# SPOKEN LANGUAGE PROCESSING

## *A Guide to* Theory, Algorithm, and System Development

- New advances in spoken language processing: *theory and practice*
- In-depth coverage of speech processing, speech recognition, speech synthesis, spoken language understanding, and speech interface design
- Many case studies from state-of-the-art systems, including examples from Microsoft®'s advanced research labs

*Spoken Language Processing* draws on the latest advances and techniques from multiple fields: computer science, electrical engineering, acoustics, linguistics, mathematics, psychology, and beyond. Starting with the fundamentals, it presents all this and more:

- Essential background on speech production and perception, probability and information theory, and pattern recognition
- Extracting information from the speech signal: useful representations and practical compression solutions
- Modern speech recognition techniques: hidden Markov models, acoustic and language modeling, improving resistance to environmental noises, search algorithms, and large vocabulary speech recognition
- Text-to-speech: analyzing documents, pitch and duration controls, trainable synthesis, and more
- Spoken language understanding: dialog management, spoken language applications, and multimodal interfaces

To illustrate the book's methods, the authors present detailed case studies based on state-of-the-art systems, including Microsoft's Whisper speech recognizer, Whistler text-to-speech system, Dr. Who dialog system, and the MiPad handheld device. Whether you're planning, designing, building, or purchasing spoken language technology, this is the state of the art—from algorithms through business productivity.

**About the Authors**

XUEDONG HUANG is founder and head of the Speech Technology Group at Microsoft Research. He received his Ph.D. from the University of Edinburgh. He is an IEEE Fellow.

ALEX ACERO and HSIAO-WUEN HON are Senior Researchers at Microsoft Research and Senior Members of IEEE. Both received doctorates from Carnegie Mellon University.

PRENTICE HALL
Upper Saddle River, NJ 07458
www.phptr.com

ISBN 0-13-022616-5
90000
9 780130 226167
V036 51010045138