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*th dialog turn.

Based on the formulation in Eq. (17.1), a dialog system is basically 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 sentence **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 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, 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.

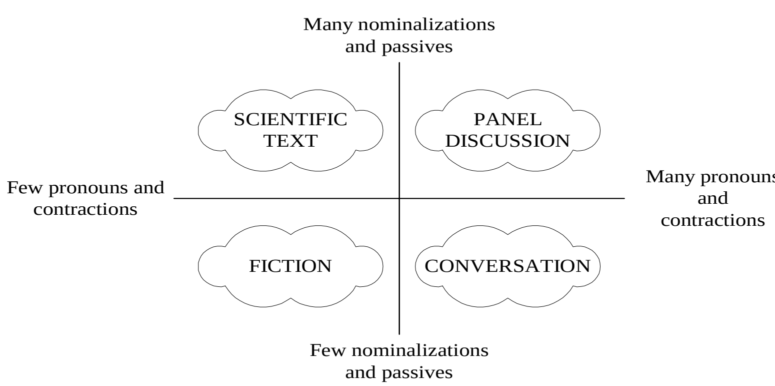

Figure 17.1 Dimensions of written vs spoken language variation.

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

[1] Nominalization is 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.*

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 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 $P(D \mid \mathbf{X})$. 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 likeliest 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.

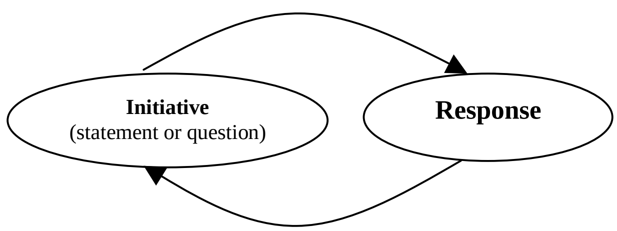

Figure 17.2 The fundamental structure of dialog: initiative and response.

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 s/he herself will do something in 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.

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.

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

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 acknowledgement* 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 dialo- 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!

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

Table 17.3 Backward looking tags.

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

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 *ans|* 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?
ans|           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 sexes 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 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.

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

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 sugoals 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 system). 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.

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

Figure 17.7 A semantic frame representation for *dog*

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.

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 A 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 hierarchy 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[3] in Dr. Who [61] are good illustration of borrowing some important concepts from object-oriented pro-

[3] The representation of semantic classes is also referred to as semantic grammars in Dr. Who.

gramming to enhance the effectiveness and efficiency of using semantic objects/classes to represent domain knowledge and linguistic expressions.

```
<!-- 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.

The semantic grammar used in the Dr Who Project [58] contains the definitions of semantic classes that refers to real-world or functional entities. A semantic class is defined as a semantic frame containing set of slots that need to be filled with terminal (verbatim) words or with recursive semantic class objects. Here *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 specify 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, an *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 an 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.

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.

```
[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.*

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.

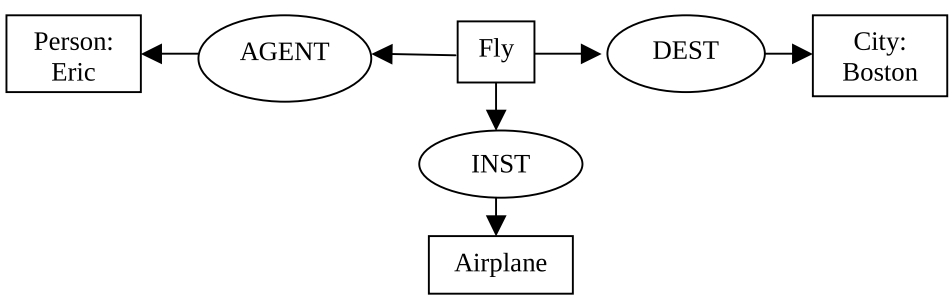

Figure 17.11 CG display form for *Lars is going to Oslo by train* [53].

Figure 17.11 illustrates a conceptual graph for the sentence *Eric is flying to Boston by a 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.

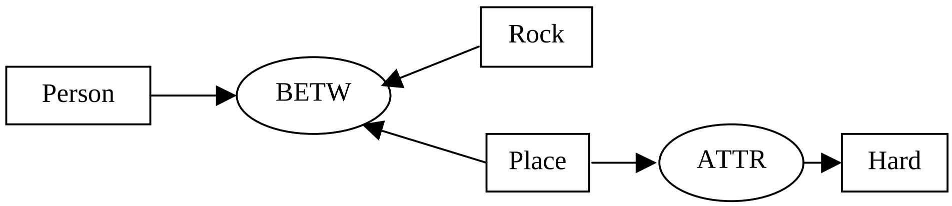

Figure 17.12 CG Display Form for *a person is between a rock and a hard place* [53].

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.

```
[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, a semantic object 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.

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.

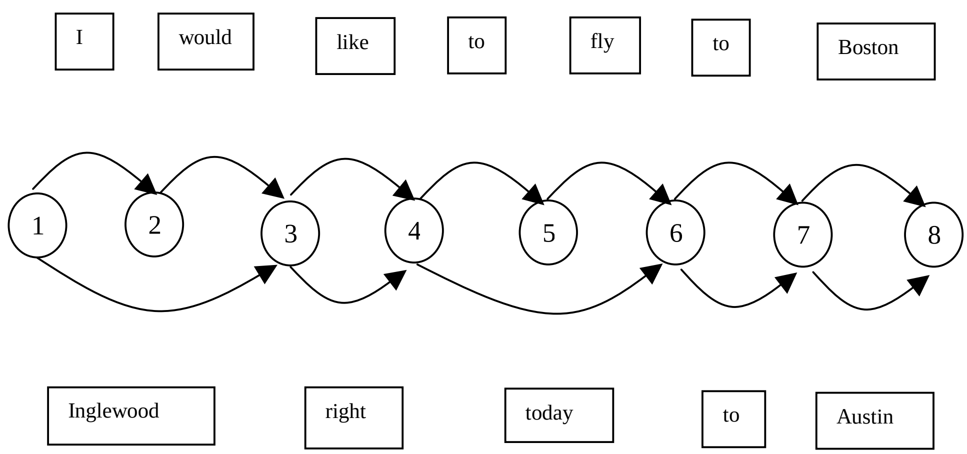

Figure 17.14 Word graph for hypotheses [61].

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.

As mentioned in Section 17.3.1, a Dr. Who grammar mostly contains 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 con-

tains 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 <u>John</u> at <u>3 PM</u>"* and "*Schedule a meeting at <u>3 PM</u> with <u>John</u>*" 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 <u>3 PM</u> would be at all possible for <u>John</u>.*"

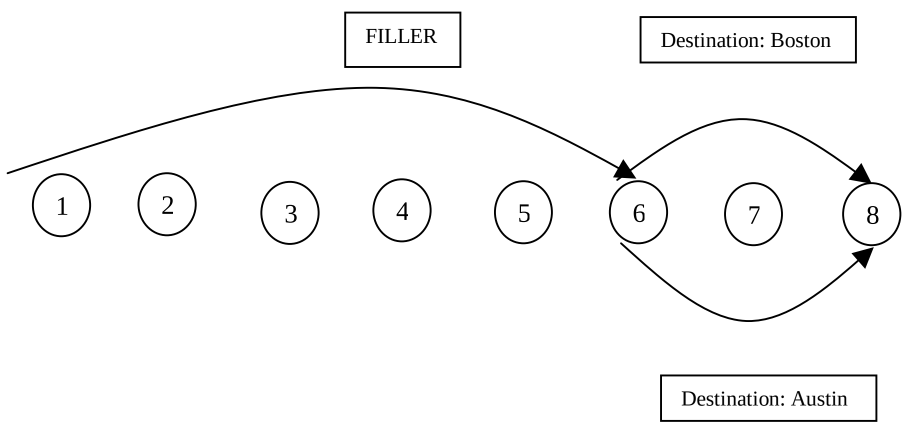

Figure 17.15 Word graph for hypotheses if only the *Destination* grammar is active [61].

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 improvements:

- 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

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

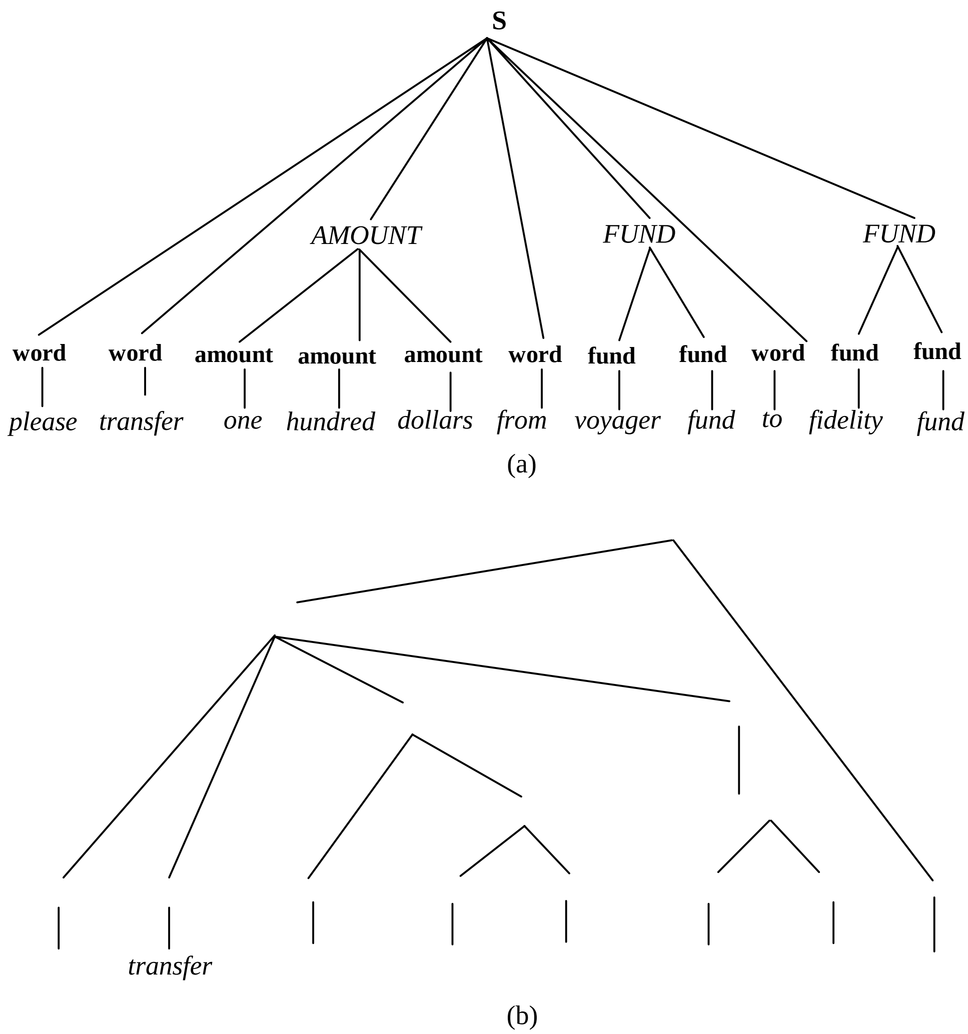

Figure 17.16 An example class tree in IBM's statistical class parser. (a) The sentence is classified into semantic classes. (b) The classed sentence is parsed into the semantic tree based on CART [15].

$P(S)$ 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; Dest_Ind ,and City, respectively. Each terminal node (like City) could be composed of a word or of a sequence of words.

```
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]..

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).

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 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 <u>her</u> 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 <u>his</u> promotion last May, John has been working very hard*" or "*<u>It</u>*

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[4] 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 Full 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 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.

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

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

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 applica-

[4] 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.

tions 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 $[\text{girl}]_i$ thought $[\text{he}]_i$ 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:

$[\text{Bill's}]_i$ photo of $[\text{him}]_i$ 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:

$[\text{Bill's}]_i$ photo of $[\text{himself}]_i$ 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 <u>Derek Acero's</u>*?" 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.

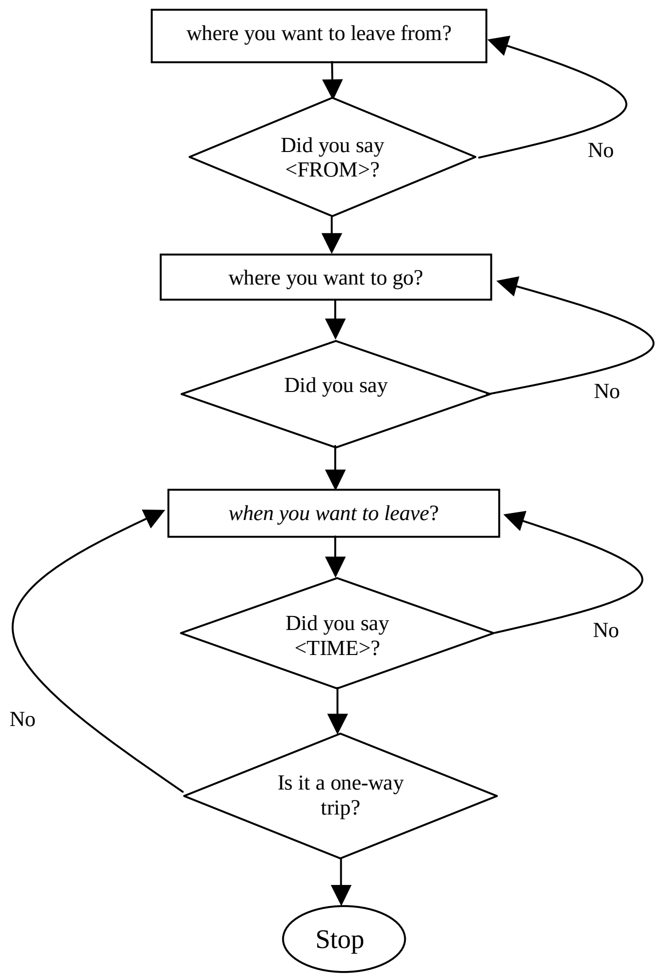

Figure 17.19 A finite state dialog grammar for airline reservation (after [19]).

In general, 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.

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.

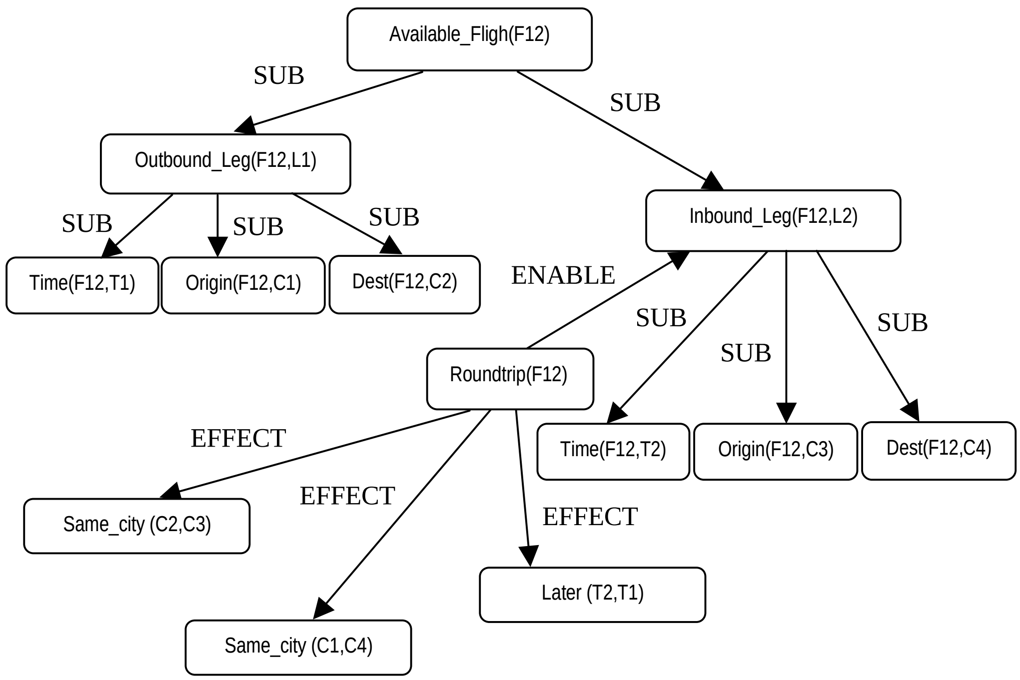

Figure 17.20 A partial plan for the airline reservation example in Figure 17.19 represented as a graph.

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.

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 departure time for the inbound leg must occur after the departure time of the outbound leg (T1 < T2). Those three inference sare 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 ϕ or ψ 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.

`I:  I would like to fly to Boston.`
`R1: Do you want to fly to Boston?` (explicit confirmation)
`R2: When do you want the flight 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 the flight to Boston*? allows the user to provide departure-time information and have a chance to confirm the destination as well.

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.

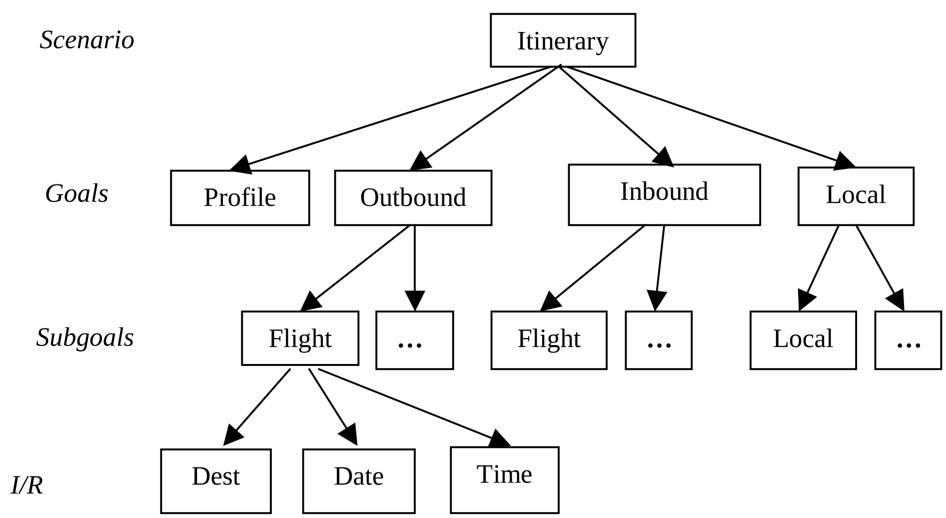

Figure 17.22 A dialog structure hierarchy for travel.

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 grammar 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 class represented as a frame, it can consult the grammar to see 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.

```
<!-- semantic class definition for ByName that has type PER-
SON 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 [lasttname]/>  />
    <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.

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.

```
<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 schedule 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.

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 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.

```
<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 flight to Boston?
Did you say the 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 mirrored 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.

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:

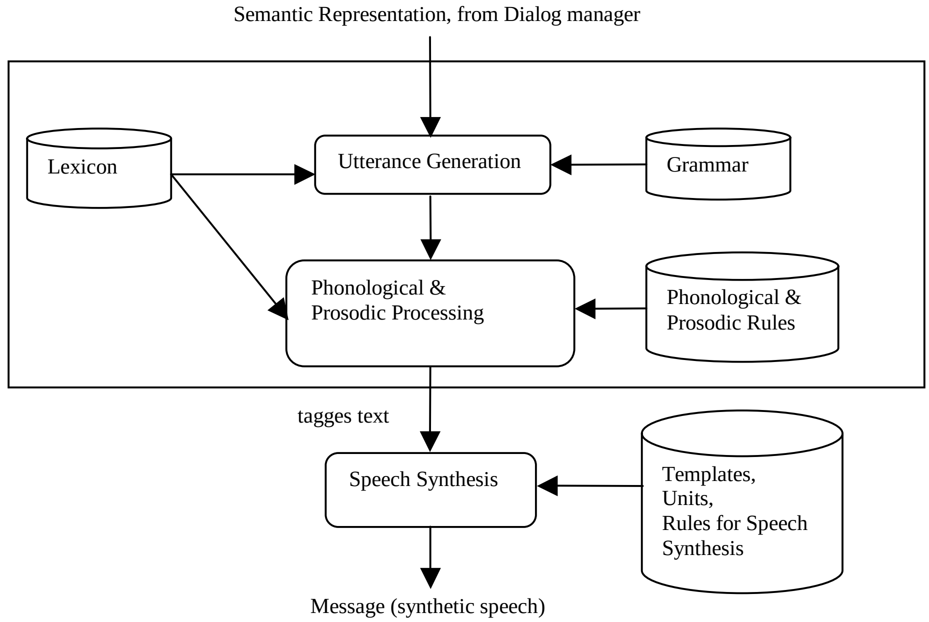

Figure 17.26 Subsystems of message generation and rendition.

- *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 establish 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 advantage 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.[5] 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:

- A—semantically independent of earlier utterances, so per-turn semantic interpretation can uniquely identify the semantic intent.

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

- 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 two 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. Moreover, the SLU systems would likely have to be tested by real subjects. Second, the evaluation of dialog behavior is highly subjective, since effectiveness and user friendliness are generally vaguely defined.

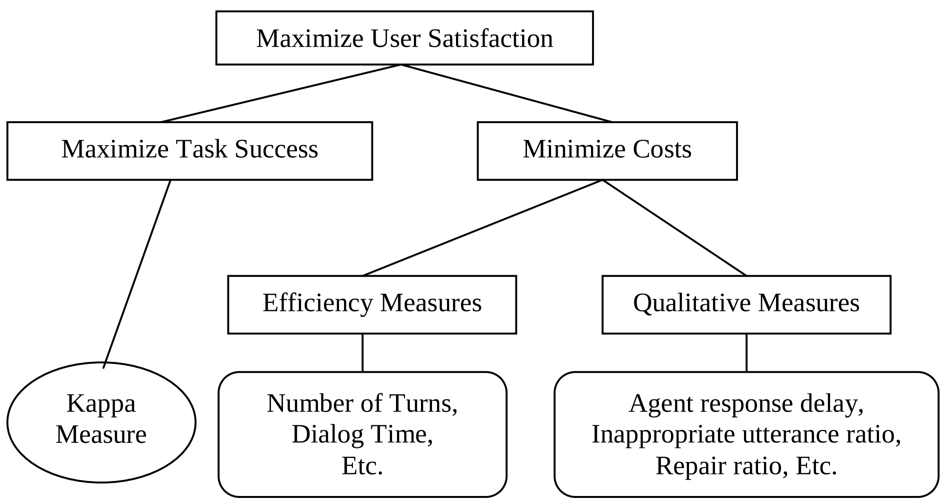

Figure 17.28 PARADISE's structure of objectives for spoken dialog performance [57].

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.

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

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:

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

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 $P(E)=\sum_{i=1}^{n}(\frac{t_i}{T})^2$, 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: $P(A)=\sum_{i=1}^{n}M(i,i)\Big/T$.

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

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 $\mathfrak{A}(x)=\dfrac{x-\overline{x}}{\sigma_x}$.

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, 58-60]. 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.

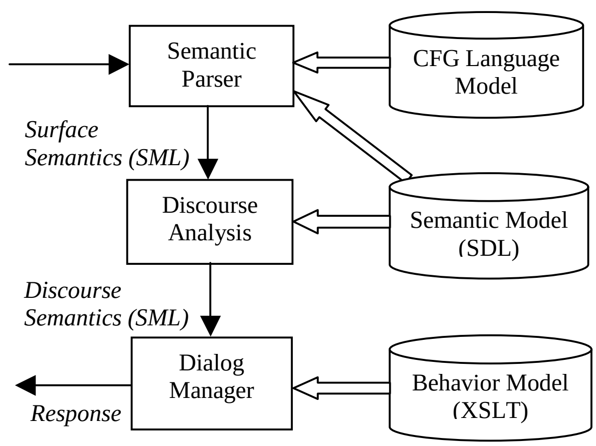

Figure 17.29 The Dr. Who system architecture [60].

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 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 Chapter 11. Once we have text transcription of user's utterances, a robust chart parser 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

```
<DirectoryQuery …>
   <PersonByName type="Person" parse="kuansan">
       Kuansan
   </PersonByName>
   <DirectoryItem type="DirectoryItem" parse="phone number">
       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 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) [62] 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].

```
<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].

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.

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.

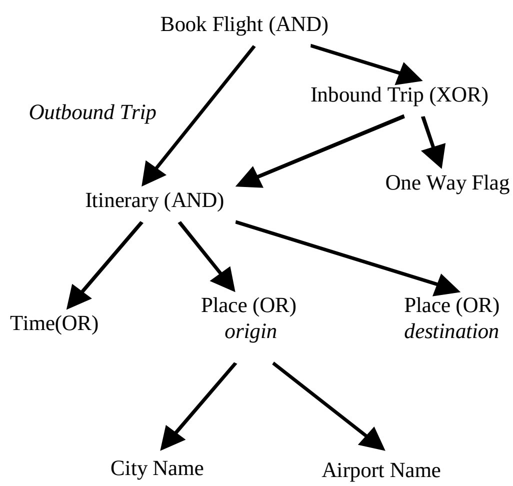

Figure 17.38 An semantic tree hierarchy corresponding to the partial plan shown in Figure 17.20 in an airline reservation application [58].

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].

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, pp. 1-38, Ablex Publishing Corporation.

[10] Cohen, P., "Models of Dialogue," *Proc.s 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, "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.D. Mori, "Dialogue Systems" in *Spoken Dialogues with Computers*, R.D. 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., Editor 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, pp. 563-582, Academic Press.

[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 od 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.