Natural Language Processing

Introductory article

James F Allen, University of Rochester, Rochester, New York, USA

CONTENTS

Natural language processing is a field that explores computational methods for interpreting and processing natural language, in either textual or spoken form.

INTRODUCTION

Computers that can speak and understand natural language have long been a key component in science fiction. Actually building machines that can understand and produce speech, however, has proven to be exceptionally difficult. While research started in the first days of functional computers in the 1950s, the complexity of the problem thwarted researchers for decades. Recently, however, significant progress has been made and, while machines are far from understanding language like humans do, useful applications involving language are now possible. We see dictation systems that allow someone to dictate a letter to a computer, internet search engines that look for pages with certain content, automated telephone systems that allow you to dial numbers and make collect calls just by speaking, and systems that can answer simple questions about the weather, traffic or stock quotes over the telephone. In addition, more sophisticated language processing systems are currently under development and will become usable in the next few years.

It is important to realize that there are several different motivations for work in natural language processing. On one side, there are engineering goals, where one is interested in finding techniques that allow computers to perform practical tasks. On the other are the cognitive science goals, where one is interested in the insight that computational models can provide for understanding human language processing. While there is overlap in research towards these goals, this article will focus mainly on issues of relevance to the cognitive science goals.

INTRODUCTION TO RELEVANT LINGUISTIC NOTIONS

The principal difficulty in processing natural language is the pervasive ambiguity that is present. Ambiguity is found at all levels of the problem. For example, all natural languages involve:

- acoustic ambiguity (given a speech signal, what words were actually spoken?);
- lexical ambiguity (e.g. 'duck' can be a noun (the animal or the cloth) or a verb (to avoid something thrown));
- structural or syntactic ambiguity (e.g. in 'I saw the man with a telescope', the telescope might be used for the viewing or might be held by the man being observed);
- semantic ambiguity (e.g. 'go' as a verb has well over ten distinct meanings in any dictionary);
- pragmatic ambiguity (e.g. 'Can you lift that rock?' may be a yes/no question or a request to lift the rock); and
- referential ambiguity (e.g. in 'Jack met Sam at the station. He was feeling ill…' it is not clear who is ill, although the remainder of the sentence might suggest a preferred interpretation).

All these forms of ambiguity may interact, producing an extremely complex interpretation process. It is the prevalence of ambiguity that distinguishes natural languages from precisely defined artificial languages such as logic and programming languages. It also makes most of the techniques developed in programming language grammars, parsing and semantics, ineffective unless significantly modified.

It will help to break apart language processing into different phases as shown in Figure 1. One side involves the understanding processes, in which we move from speech to intentions. The other side involves generation, moving from intentions to speech. The stages of understanding language consist of transforming sound into words (speech recognition), words into meaning (understanding), and meaning into knowledge, intention, and action

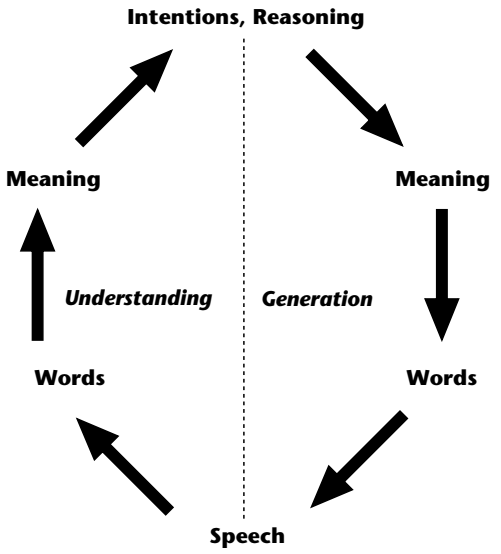

Figure 1. The stages of natural language processing.

(pragmatics and reasoning). For instance, say someone comes up and speaks to you. Using speech recognition you identify the words as *Do you know the time?* Using understanding techniques, you identify the meaning as a question about whether you know the time. Finally, using pragmatics and reasoning you decide that the stranger wants you to tell him the time and decide to answer the question. The stages of language generation account for the opposite process of generating a response. You decide to respond and form an intention to tell him the time, this is mapped into a meaning (content planning), then meaning to words, say, 'It's three p.m.' (generation), and finally words to sound (speech synthesis).

Each of these steps will be considered below. For many applications, we need only one or two stages. For instance, current dictation systems do only the first phase, mapping speech to words, and do not interpret the words any further.

SPEECH RECOGNITION: SOUNDS TO WORDS

While speech recognition systems are in use in practical applications now, they work well only in some specific situations. The important factors affecting the performance of recognition systems include:

1. the size of the vocabulary to be recognized,
2. the range of speakers that need to be handled,
3. the quality of the sound capture devices (e.g. microphones and recording), and
4. the style of the speech itself.

In general, the smaller the vocabulary, the smaller the number of speakers, the better the sound capture, and the more controlled the style of the speech, the better the recognition is. In practice, however, we usually have to balance one aspect against another to get acceptable performance in a specific application. For example, the systems that allow you to make long distance calls entirely by voice must handle hundreds of thousands of customers. They compensate for the problems caused by the large number of speakers by using a very small vocabulary (e.g., just digits and words such as 'yes' and 'no'), and by constraining the style of the speech by giving specific instructions to the person about what to say when. By doing this, the applications are usable and are saving telephone companies millions of dollars a year in operation costs. A very different case involves dictation systems. These must handle large vocabularies, say 40 000 to 100 000 words. To compensate for this, they are trained for a single speaker using a microphone held close to the mouth, and require people to speak carefully. Under these conditions, such systems can attain 95% word accuracy or better for many users.

As we start to relax the constraints more, accuracy starts to plummet. For example, currently the best speech recognition systems, when trying to recognize spontaneous speech in real time between two people having a conversation, can recognize only half of the words correctly.

Most speech recognition systems use the same basic set of techniques. The acoustic signal is digitized and converted into a sequence of sound frequency–intensity plots covering small segments of the speech. These plots are used to extract out a vector of measurements, such as intensity in different frequency bands, changes of intensity in different frequency bands from one window to the next, and the acceleration of changes of intensity in the different frequency bands. The vectors are then classified into categories based on their similarities to 'prototype' vectors. Typically, systems use 256 or 512 different categories, which comprise the *codebook*. At this stage, the signal has been converted into a sequence of symbols from the codebook.

The next phase of analysis uses models of the relevant linguistic events you want to recognize, such as words or phonemes. For large vocabulary recognition, word-based models are too hard to construct, and phoneme-based models are too inaccurate. As a result, most large vocabulary recognition systems use subword models, an intermediate level about the size of syllables, larger than the phoneme, but smaller than a word. Each

subword is represented by a Hidden Markov Model (HMM), from which one can compute the probability that a certain sequence of codebook symbols might have been generated in producing that subword. To perform speech recognition on an input signal represented as a sequence of codebook symbols, one simply finds the sequence of subword that maximizes the probability of producing that input signal.

The power of HMMs is that there are known effective algorithms for estimating the transition probabilities from a corpus of speech data, and for finding the path through a given HMM model that maximizes the probability of observing the input (the Viterbi algorithm).

So far, the description of the system has considered acoustic information only. Using only acoustic information, speech recognizers do not perform well. They must also use higher level linguistic information to predict what word sequences are likely to occur. This is called the language model. There are two types of language models in common use. The first is called an *n-gram model*, which is a probability model that specifies the probability of a given word w following a sequence of the previous $n-1$ words. A 2-gram (or bigram) model, for instance, simply provides the probability that the word w will follow the previous word. Such n-gram models are built by estimating the probability distribution from a large corpus of transcripts of speech. In an ideal world with unlimited data, one would use large amounts of previous context. In practice, however, speech recognition systems typically only use bigram or trigram models. Given an n-gram model, one can construct an HMM and use the training and search algorithms developed for HMMs to drive the recognition process. The other type of language model is the *probabilistic finite-state machine*. This is simply a probabilistic graph that specifies all possible sequences of words that can occur in interactions. This technique is common in practical applications where the vocabulary and range of language is not large.

APPROACHES TO LANGUAGE PROCESSING

As we consider the other levels of language processing, especially language understanding, we have a problem. With speech recognition, we know what the desired result is, namely what words were spoken. But we do not have such a clearly defined notion of what meaning and understanding are. This means it is hard to evaluate how well we are doing. The classic answer to this problem has been the Turing test. A system would be said to understand if a set of human judges could not distinguish a machine from a human by asking any questions they wished for as long as they wished. The problem with this test is that it is an all-or-nothing evaluation. It is hard to use this method to help drive incremental progress since we are so far from the goal. Today, most researchers in the field take a pragmatic approach and consider how well language systems can perform a specific task. For example, say the task is to find relevant webpages given natural language input such as the sentence 'I want to find sources that deal with economic growth in the late nineteenth century'. For such an application, we evaluate how well it does the task, say by checking how many relevant pages it finds (the recall) and what proportion of the pages found are relevant (the precision).

In the early decades of research (1960–1990), it was believed that machines would have to acquire a capability for 'deep' understanding of language before the technology could be useful. Deep understanding involves constructing a precise specification of the meaning of the sentence and being able to perform reasoning on the information. In the 1990s, however, researchers found that 'shallow' statistical analysis techniques that capture only some structure and content of language could be useful in a wide range of applications. For example, most webpage search engines just look for documents that contain the words or short phrases found in the query. While such techniques produce robust systems with reasonable recall, the precision is often a problem. You may find that the system returns so many irrelevant webpages that finding the appropriate ones can be very difficult. In addition to information retrieval, there are many practical applications, which are becoming feasible using statistical approaches, that fall far short of 'deep' understanding including information extraction (e.g., generating a database of merger and acquisitions activity from newspaper reports), information access (e.g., using speech over the telephone to find out information on airline schedules), rough machine translation (e.g., automatically translating equipment maintenance manuals), and writing tools (e.g., spelling correction and grammar checking tools in word processors).

For some period of time, many researchers viewed the deep and statistical processing techniques to be in opposition to each other. More recently, most researchers realize that each area of work addresses different aspects of the problem

and that both are needed for long-term success in the field.

COMPUTATIONAL PHONOLOGY AND MORPHOLOGY

Language is made of words, either realized as sound or in written form. Sometimes words are considered atomic elements of language, but they actually have a rich structure. Many languages use a wide variety of word forms to indicate how words relate to each other in the sentence. In English, we see word forms to encode number and person agreement between pronouns and verbs (*I am, you are, she is, they are, …*), grammatical role in pronouns (*he, his, him*), tense and aspect (*go, went, gone, going*), parts of speech (*destroy, destruction*, *destroyable, destructive, destructively*) and semantically related words (*untighten, retighten, pretighten, overtighten*). The study of the structure of words and how different forms relate to each other is called morphology, and the development of algorithms for processing word forms is computational morphology.

A very useful computational technique is called *lemmatizing*, which identifies the root form (or lemma) and *affixes* of a word for further processing. For example, such a system might break apart the words weave, wove, woven, and weaving into a root form *weave* plus features such as *present, past, pastparticiple*, and *presentparticiple*. Stemming is important as a first pass for producing deeper structural and semantic analyses of sentences as the lemmas typically identify key components of the semantic content. Another application of stemming is in information retrieval, where it would be useful for all the forms of *weave* to map to the same lemma. Thus if you look for documents that include the word *weave*, the program would also find documents that contain the word *wove*.

One of the most influential techniques for morphological analysis uses finite-state transducers (FSTs), which are finite-state machines that produce an output for every input. Such a device would take a word like *happier* as input and produce the output *happy* + *er* using an FST that maps the first four letters to themselves, then maps the *i* to a *y*, inserts a space, and maps the last two letters, *e* and *r* to themselves. One of the most common frameworks for morphological analysis uses a set of FSTs that are run in parallel; all of them must simultaneously accept the input and agree on the output.

Phonology studies the relationship between words and how they are realized in speech. The idealized sounds of language are captured by an alphabet of *phonemes*, each phoneme representing an abstraction of a primitive meaningful sound in the language. In actual fact, the same phoneme can be pronounced very differently in different contexts. These variants within a single phoneme are called *allophones*. Computational models of phonology are important in both speech synthesis and speech recognition work. It is not feasible, for instance, for a speech synthesizer that can read books to have a mapping of all words to their pronunciation. Rather, it often has to produce a phonetic spelling from the orthographic spelling of the word. In general, the same computational techniques used for morphological processing are also used for phonological processing.

PARSING

The process of parsing involves mapping from a sequence of words to an internal representation. A *syntactic* parser takes a grammar and a word sequence, and finds one or more combinations of rules in the grammar that could produce the word sequence. The output then is a parse tree that shows the different phrasal constituents that make up the sentence. A *semantic* parser maps a word sequence to a representation of its meaning in some formal meaning representation language. Most modern grammars and parsers combine these two tasks and construct a syntactic structure and a meaning representation simultaneously using *rule-by-rule compositional semantics*. In such an approach, the rules in the grammar specify not only structural relations (e.g., a noun phrase can consist of an article ('A') followed by a noun ('dog')), but also a fragment of the meaning representation, say, in some form of logic. As rules combine smaller phrases into larger ones, the meaning fragments of the smaller phrases are combined into larger fragments for the entire phrase. It is possible to separate these processing phases and use a grammar that produces a solely syntactic structure, and then define another set of rules for mapping the syntactic structures into meanings. But so far researchers have found little advantage in doing so. It is also possible to forgo syntactic processing altogether, and attempt to build a meaning representation directly from the word sequence. Such approaches are typically only used in quite limited applications in which the sentences remain fairly simple, and the meaning representation is specialized to the particular application and lacks generality.

The most common grammatical formalisms used in computational systems are augmented

context-free grammars. Such grammars have a core context-free grammar, but each rule is associated with a set of features that capture grammatical phenomena such as number agreement and encode semantic information. Most of these systems use *feature unification* and can use quite elaborate typed feature structure systems. For example, the rule

```
NP[AGR ?x] -> ART[AGR ?x] N[AGR ?x]
```

states that a noun phrase (NP) can consist of an article (ART) followed by a noun (N), but only the ART and N have compatible agreement feature (AGR) values. Furthermore, the NP acquires the same agreement feature. For example, the word *the* allows both third person singular and plural forms, so its AGR feature would be a set consisting of 3s (third person singular) and 3p (third person plural). The word *dog*, on the other hand, allows only third person singular, so the AGR feature would be the singleton set {3s}. Feature unification is like set intersection, so these two features can be unified and the result is {3s}. This becomes the AGR value of the newly introduced NP constituent. Likewise, agreement would be enforced between the subject and the verb by the S rule (a rule for sentences) that combines an NP followed by a verb phrase (VP). Note that with the AGR feature, the grammar will not accept sentences such as *A dogs barked* or *The dogs barks* even though the word sequences are fine based simply on the basic word categories. Figure 2 shows a parse tree that is the analysis of the sentence *The dog barked*, showing the basic categories and the AGR feature.

Context-free grammars augmented with feature unification provide the power to express much of natural language in fairly concise ways, and have become the most common grammatical formalism for computational grammars. They have also provided the formal underpinnings for linguistic theories such as head-driven phrase structure grammar (HPSG) and lexical-functional grammar (LFG).

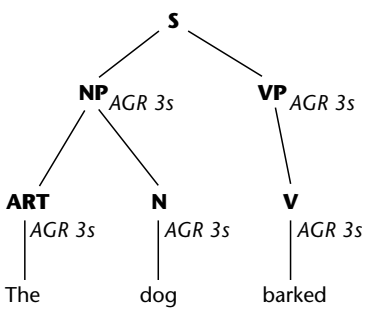

Figure 2. A parse tree.

A parser performs a search process through the possible combinations of grammatical rules to find trees that can account for the input. Most modern parsing systems use some form of chart parsing. A *chart* is a data structure that keeps track of the work done so far in parsing: both the possible constituents that have been found so far, and the rules that have partially matched so far. The parser then runs by constantly updating the chart with new information. There are two properties of chart parsers that make them efficient. The first is that the parser never does the same work twice. Before doing any work, it checks the chart to see whether the work has been done before. The second property is that information is shared across different possible interpretations. For example, a particular noun phrase might be able to be used in a number of different overlap interpretations. That noun phrase, however, appears only once in the chart.

SEMANTICS: LEXICAL AND COMPOSITIONAL

Semantics concerns the meaning of sentences. We need to define a language in which to express that meaning, and we need some method of computing the meaning from the parse tree. To start this process, we need to associate meanings with words. Words typically have many senses (e.g., 'seal' can denote an animal, a gasket for making containers airtight, a symbol representing some authority, or an action of closing an envelope). Such meanings can often be arranged hierarchically according to their properties. For example, a 'seal' is a particular type of animal, which is a living thing, which is a physical object. The properties of being a seal arise from particular properties of seals (e.g., seals bark), from more general properties (e.g., animals eat, physical objects have weight). Such representations are often called *semantic networks*. These properties are important for helping determine the appropriate sense while parsing. For instance, if we have a sentence 'The seal ate a fish', then we can conclude that we are talking about a seal as an animal, and not the seal of a canning jar, or the symbol of the president of the United States.

Other words tend to have more complex semantic structures. A verb, for instance, typically requires a set of arguments to complete its meaning. For instance, the verb 'put' describes some sort of physical action, which is determined once we know who did the action, what was acted upon, and where it went. In many computational representations a complex object is defined as a *type* and a set of *roles*. The roles for the PUT action might

be called AGENT, THEME, and TO-LOC. Such representations have been proposed in philosophy by Davidson, in linguistics by Fillmore, and are often called *frames* in computational work. Expressed in first-order predicate calculus, the meaning of the sentence *Jack put the book in the box* might be

```
∃e . PutEvent(e) & AGENT(e, J1)
  & THEME(e, B1) & TO-LOC (e, B2)
```

where the constants J1, B1, and B2 are defined elsewhere and represent Jack, the book, and the box. In a frame-like knowledge representation, this might be represented as

```
[PUTEVENT e
  [AGENT J1]
  [THEME B1]
  [TO-LOC B2]]
```

but the semantics remains the same.

Compositional Semantics

One of the key ideas used in many modern parsers is compositional semantics, where the semantic form of the sentence is constructed simultaneously with the syntactic form. Each rule in the grammar specifies both syntactic and semantic operations. To make this work, systems use either feature unification or lambda abstraction. For instance, the meaning of a VP 'ate the pizza' might be

```
λ x . [EAT [AGENT x] [THEME P1]]
```

which is a function that takes an argument and places it the AGENT slot. With this interpretation, the semantic component of a grammatical rule for declarative sentences, namely

```
S -> NP VP
```

would apply the meaning of the VP (like *ate the pizza* above) to the meaning of the NP to produce the meaning of the sentence. While the lambda calculus provides a nice formal model for compositional semantics, many parsers do the equivalent thing using feature unification. For example, we could write the same rule as

```
S[LF ?lf] ->
  NP[LF ?subj] VP[SUBJ ?subj LF ?lf]
```

Here, the VP rule would produce a constituent of the form

```
VP[SUBJ ?s LF [EAT [AGENT ?s] [THEME
  P1]]
```

and when the S rule is applied, the *?subj* variable will be bound to the meaning of the subject NP and the LF feature of the new S constituent would be bound to the completed LF in the VP.

Robust Interpretation

In many applications, it is not feasible to require a full syntactic analysis of each sentence. For example, in a spoken dialogue system, speech recognition errors may prevent the parser ever receiving all the correct words spoken. In a system that retrieves relevant articles from a newspaper database, we can't expect to have a grammar that has full grammatical coverage. In these cases, it is important to be able to extract out fragments of meaning from the input. Using a bottom-up chart parser provides one obvious approach. Even if a full syntactic analysis is not found, the chart will contain analyses of parts of the input – the major noun phrases, for instance. And given these analyses it may be possible to infer the intended content of the entire utterance. For example, consider a system that interacts with a user in spoken language to identify good routes for trains. Say the conversation has concerned a particular train TR1. The next utterance is *now send it on to Avon via Bath*, but say the speech recognition output is 'NOISE ENDED TO AVON VIA BATH'. From the fragments, one can recognize a path (to Avon via Bath), and from the discourse context, one knows the topic was train TR1. Thus, it is likely that the speaker intended to send TR1 from Avon to Bath.

For applications such as information retrieval and information extraction from textual databases, most systems forgo full parsing altogether and depend entirely on pattern-based techniques that look for patterns in the input that relate to information of concern.

PRAGMATICS: SPEECH ACTS AND DISCOURSE

Applications that involve interaction in natural language require significant processing beyond deriving the semantic interpretation of an utterance. In everyday language, a conversational system needs to recognize what the intentions of the user are in order to respond appropriately. For instance, a sentence such as *Do you know the time* might be a question about your state of knowledge, a request that you tell the speaker the time, or a reminder that it is late. Each of these different acts is called a *speech act*. Typically, the linguistic structure of the utterance does not identify which speech act has been performed. We need to consider the larger context of the conversation, namely the current situation and

what we know of the user's plans and goals, in order to identify the correct speech act.

The context of a conversation, or discourse in general, is captured by several different components. Two of the most important are the attentional state and the intentional state. The attentional state includes the recent history of the conversation, especially information about what was said in the last utterance. The objects mentioned in the last utterance are the most likely candidates for pronominal reference in the current sentence. In addition, the structure of the last sentence is needed to interpret forms of ellipsis (e.g., *I went to the store. John did too*.) One of the more influential models for pronoun interpretation is *centering theory*. This model places a high reliance on the objects mentioned in the immediately previous sentence, and distinguishes one object as most relevant.

The intentional state captures the motivations of the speakers engaged in the conversation. In general, computational work has focused on practical, or task-oriented, dialogue in which the participants are speaking to each other in order to perform a certain task, be it obtaining information, learning mathematics, getting help on home repair, or designing kitchens. In all these domains, we can represent the reasoning processes of the participants as a form of *planning*. The study of planning has a long history in artificial intelligence and is typically formalized as a search process through possible future actions. Actions are represented as operators that change the world, and are typically described in terms of their preconditions (what must be true when the action is attempted) and effects (what will be true after it is successfully executed). The process of planning starts with a library of actions, an initial world state, a statement of the goal conditions, and searches for a sequence of actions that can be executed starting in the initial state and resulting in a state that satisfies the goal conditions. By modeling the participants' planning processes, we can interpret their utterances by finding out how their utterances further their goals. These techniques can be used to address many problems, including word sense disambiguation (which reading makes sense in the plan), pronoun reference (which objects are most likely to be used in the plan), and most importantly, what speech act is intended. To do this last task we must model speech acts as operators in a planning system. The effects of a speech act are changes in the speaker's and hearer's mental state, namely their beliefs and goals. For example, an effect of a REQUEST act is that the hearer knows the speaker wants to get the hearer to do the requested act. By considering each possible speech act and how their effects might fit into the current task, we can determine which interpretation makes the most sense in context (e.g., a process called *plan inference*). A simple plan-based conversational agent uses plan inference to identify plausible goals of the user, and then uses planning to plan a response that is most helpful given the current task.

GENERATION: MEANING TO WORDS

The generation side of natural language processing has the same distinctions and levels of processing as the interpretation side. It is often viewed as a planning problem, where the input is a set of communicative goals and the output is a series of utterances that realize those goals. The process is typically broken into at least two levels: content planning, in which the system decides *what* needs to be communicated, and surface generation, in which the system determines the details of *how* it is to be communicated.

Content planning can roughly be thought of as mapping from abstract communicative goals (e.g., describe how to find the car in the parking lot) down to a sequence of specific speech acts (e.g., REQUEST that the person first go to the library entrance, then …). Rather than planning from first principles, generation is usually driven by larger scale *schemas* that relate conversational goals and specific circumstances to particular strategies for attaining those goals. By repeatedly refining conversational goals to more specific levels, one eventually ends up at the concrete speech act level. In addition to determining the speech, one must also plan the content of the act. A particular issue is determining what content to mention in order to produce a successful referring expression. In general, one is looking for a small set of properties that will successfully distinguish the intended object from other competitors in the context. For instance, in a setting in which there is a large red ball (B1), a large blue ball (B2), and a small red ball (B3), one might refer to B2 as 'the blue ball', not mentioning size, and B3 as 'the small ball', not mentioning color.

Surface generation can be viewed as the reverse process of parsing. It is given a logical form, the same or similar to the meanings produced by a parser. The challenge is to capture a given meaning in a natural sounding sentence of extended discourse. Some grammatical frameworks are designed in a way that they can be used both for parsing and for generation. Often, however, even when this is the case, different grammars are used.

One reason is that that there is a different set of concerns facing generation. For example, consider the passive construct (e.g., *John baked a cake* vs. *A cake was baked by John)*. A grammar for parsing might simply have rules that map both these sentences to the same logical form. A generator, on the other hand, must make a decision about which to use, and picking the wrong choice can lead to clumsy interactions or utterances that produce incorrect implications (e.g., consider *A cake was baked by John* as the answer to the question *What did John cook?*). In addition, the surface generator may have to make choices between different lexical realizations (e.g., *John donated $5 million to the museum* vs. *John gave the museum a $5 million contribution*).

CONCLUSION

Natural language processing (NLP) technology is becoming an area of great practical importance and commercial application. Within the foreseeable future, NLP will revolutionize the way we use and think about computers in a profound way. On the theoretical side, as the computational models become more sophisticated, we can expect new insights into possible models of human processing that can then become the subject of empirical experimentation.

Further Reading

Allen JF (1995) *Natural Language Understanding*. Redwood City, CA: Benjamin Cummings.

Allen JF, Miller B, Ringger E and Sikorski T (1996) *A Robust System for Natural Spoken Dialog*. Paper presented at the 31st Meeting of the Association for Computational Linguistics, Santa Cruz, CA.

Carpenter B (1992) *The Logic of Typed Feature Structures*. Cambridge, UK: Cambridge University Press.

Davidson D (1967) The Logical Form of Action Sentences. In: Rescher N (ed.), *The Logic of Decision and Action*. Pittsburgh, PA: University of Pittsburgh Press.

Grosz BJ and Sidner CL (1986) Attention, Intentions, and the Structure of Discourse. *Computational Linguistics* **12**(3): 175–204.

Hobbs JR, Appelt D, Bear J *et al*. (1996) FASTUS: A cascaded finite-state transducer for extracting information from natural-language text. In: Roche E and Schabes Y (eds) *Finite-state Devices for Natural Language Processing*, pp. 383–406. Cambridge, MA: MIT Press.

Jurafsky D and Martin JH (2000) *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics and Speech Recognition*. Englewood Cliffs, NJ: Prentice-Hall.

Manning C and Schutze H (1999) *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

Pollard C (1994) *Head-driven Phrase Structure Grammar*. Chicago, IL: Chicago University Press.

Rabiner L and Juang BH (1993) *Fundamentals of Speech Recognition*. Englewood Cliffs, NJ: Prentice-Hall.

Reiter E and Dale R (2000) *Building Natural Language Generation Systems*. Cambridge, UK: Cambridge University Press.

Sproat R (1993) *Morphology and Computation*. Cambridge, MA: MIT Press.