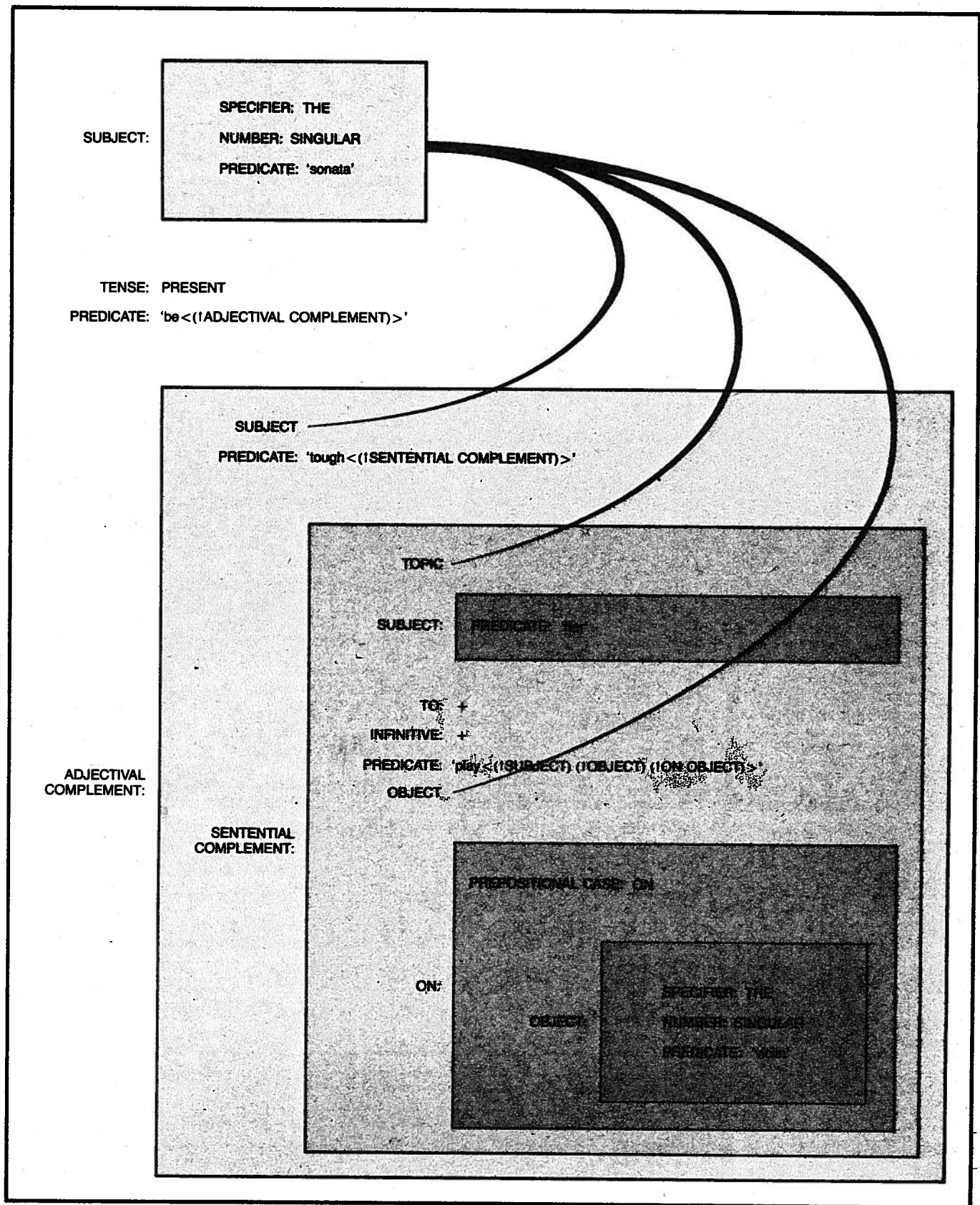


an Article from

**SCIENTIFIC  
AMERICAN**

SEPTEMBER, 1984 VOL. 251, NO. 3

The sonata is tough for her to play on the violin.



**REPRESENTATION OF A SENTENCE** in a way that makes explicit the linguistic relations among its parts has been a goal of the science of linguistics; it is also a necessary aspect of the effort to design computer software that "understands" language, or at any rate can draw inferences from linguistic input. In this illustration a sentence is given in "functional structure" form, which has the property that

when part of a sentence plays a role in another part, the former is "nested" in the latter. The nesting is shown by placing one box in another, or (in three places) by a "string." The sentence was analyzed by Ronald M. Kaplan and Joan Bresnan of Stanford University and the Xerox Corporation's Palo Alto Research Center. Another functional-structure diagram appears in the illustration on pages 142-143.

# Computer Software for Working with Language

*Programs can manipulate linguistic symbols with great facility, as in word-processing software, but attempts to have computers deal with meaning are vexed by ambiguity in human languages*

by Terry Winograd

In the popular mythology the computer is a mathematics machine: it is designed to do numerical calculations. Yet it is really a language machine: its fundamental power lies in its ability to manipulate linguistic tokens—symbols to which meaning has been assigned. Indeed, “natural language” (the language people speak and write, as distinguished from the “artificial” languages in which computer programs are written) is central to computer science. Much of the earliest work in the field was aimed at breaking military codes, and in the 1950’s efforts to have computers translate text from one natural language into another led to crucial advances, even though the goal itself was not achieved. Work continues on the still more ambitious project of making natural language a medium in which to communicate with computers.

Today investigators are developing unified theories of computation that embrace both natural and artificial languages. Here I shall concentrate on the former, that is, on the language of everyday human communication. Within that realm there is a vast range of software to be considered. Some of it is mundane and successful. A multitude of microcomputers have invaded homes, offices and schools, and most of them are used at least in part for “word processing.” Other applications are speculative and far from realization. Science fiction is populated by robots that converse as if they were human, with barely a mechanical tinge to their voice. Real attempts to get computers to converse have run up against great difficulties, and the best of the laboratory prototypes are still a pale reflection of the linguistic competence of the average child.

The range of computer software for processing language precludes a comprehensive survey; instead I shall look at four types of program. The programs deal with machine translation, with word processing, with question an-

swering and with the adjuncts to electronic mail known as coordination systems. In each case the key to what is possible lies in analyzing the nature of linguistic competence and how that competence is related to the formal rule structures that are the theoretical basis of all computer software.

The prospect that text might be translated by a computer arose well before commercial computers were first manufactured. In 1949, when the few working computers were all in military laboratories, the mathematician Warren Weaver, one of the pioneers of communication theory, pointed out that the techniques developed for code breaking might be applicable to machine translation.

At first the task appears to be straightforward. Given a sentence in a source language, two basic operations yield the corresponding sentence in a target language. First the individual words are replaced by their translations; then the translated words are reordered and adjusted in detail. Take the translation of “Did you see a white cow?” into the Spanish “¿Viste una vaca blanca?” First one needs to know the word correspondences: “vaca” for “cow” and so on. Then one needs to know the structural details of Spanish. The words “did” and “you” are not translated directly but are expressed through the form of the verb “viste.” The adjective “blanca” follows the noun instead of preceding it as it does in English. Finally, “una” and “blanca” are in the feminine form corresponding to “vaca.” Much of the early study of machine translation dwelt on the technical problem of putting a large dictionary into computer storage and empowering the computer to search efficiently in it. Meanwhile the software for dealing with grammar was based on the then current theories of the structure of language, augmented by rough-and-ready rules.

The programs yielded translations so bad that they were incomprehensible. The problem is that natural language does not embody meaning in the same way that a cryptographic code embodies a message. The meaning of a sentence in a natural language is dependent not only on the form of the sentence but also on the context. One can see this most clearly through examples of ambiguity.

In the simplest form of ambiguity, known as lexical ambiguity, a single word has more than one possible meaning. Thus “Stay away from the bank” might be advice to an investor or to a child too close to a river. In translating it into Spanish one would need to choose between “*orilla*” and “*banco*,” and nothing in the sentence itself reveals which is intended. Attempts to deal with lexical ambiguity in translation software have included the insertion of all the possibilities into the translated text and the statistical analysis of the source text in an effort to decide which translation is appropriate. For example, “*orilla*” is likely to be the correct choice if words related to rivers and water are nearby in the source text. The first strategy leads to complex, unreadable text; the second yields the correct choice in many cases but the wrong one in many others.

In structural ambiguity the problem goes beyond a single word. Consider the sentence “He saw that gasoline can explode.” It has two interpretations based on quite different uses of “that” and “can.” Hence the sentence has two possible grammatical structures, and the translator must choose between them [see bottom illustration on page 133].

An ambiguity of “deep structure” is subtler still: two readings of a sentence can have the same apparent grammatical structure but nonetheless differ in meaning. “The chickens are ready to eat” implies that something is about to eat something, but which are the chickens? One of the advances in linguistic

theory since the 1950's has been the development of a formalism in which the deep structure of language can be represented, but the formalism is of little help in deducing the intended deep structure of a particular sentence.

A fourth kind of ambiguity—semantic ambiguity—results when a phrase can play different roles in the overall meaning of a sentence. The sentence "David wants to marry a Norwegian" is an example. In one meaning of the sentence the phrase "a Norwegian" is referential. David intends to marry a particular person, and the speaker of the sentence has chosen an attribute of the person—her being from Norway—in order to describe her. In another meaning of the sentence the phrase is attributive. Neither David nor the speaker has a particular person in mind; the sentence simply means that David hopes to marry someone of Norwegian nationality.

A fifth kind of ambiguity might be called pragmatic ambiguity. It arises from the use of pronouns and special nouns such as "one" and "another." Take the sentence "When a bright moon ends a dark day, a brighter one will follow." A brighter day or a brighter moon? At times it is possible for translation software to simply translate the ambiguous pronoun or noun, thereby preserving the ambiguity in the translation. In many cases, however, this strategy is not available. In a Spanish translation of "She dropped the plate on the table and broke it," one must choose either the masculine "lo" or the feminine "la" to render "it." The choice forces the translator to decide whether the masculine "plato" (plate) or the feminine "mesa" (table) was broken.

In many ambiguous sentences the meaning is obvious to a human reader,

but only because the reader brings to the task an understanding of context. Thus "The porridge is ready to eat" is unambiguous because one knows porridge is inanimate. "There's a man in the room with a green hat on" is unambiguous because one knows rooms do not wear hats. Without such knowledge virtually any sentence is ambiguous.

Although fully automatic, high-quality machine translation is not feasible, software is available to facilitate translation. One example is the computerization of translation aids such as dictionaries and phrase books. These vary from elaborate systems meant for technical translators, in which the function of "looking a word up" is made a part of a multilingual word-processing program, to hand-held computerized libraries of phrases for use by tourists. Another strategy is to process text by hand to make it suitable for machine translation. A person working as a "pre-editor" takes a text in the source language and creates a second text, still in the source language, that is simplified in ways facilitating machine translation. Words with multiple meanings can be eliminated, along with grammatical constructions that complicate syntactic analysis. Conjunctions that cause ambiguity can be suppressed, or the ambiguity can be resolved by inserting special punctuation, as in "the [old men] and [women]." After the machine translation a "post-editor" can check for blunders and smooth the translated text.

The effort is sometimes cost-effective. In the first place, the pre-editor and post-editor need not be bilingual, as a translator would have to be. Then too, if a single text (say an instruction manual) is to be translated into several languages, a

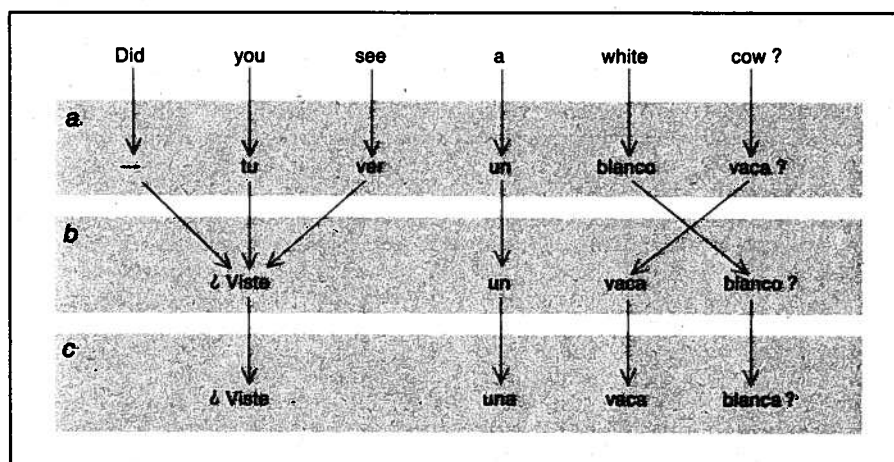
large investment in pre-editing may be justified because it will serve for all the translations. If the author of the text can be taught the less ambiguous form of the source language, no pre-editor is needed. Finally, software can help in checking the pre-edited text to make certain it meets the specifications for input to the translation system (although this is no guarantee that the translation will be acceptable).

A machine-translation system employing pre- and post-editing has been in use since 1980 at the Pan-American Health Organization, where it has translated more than a million words of text from Spanish into English. A new system is being developed for the European Economic Community, with the goal of translating documents among the official languages of the community: Danish, Dutch, English, French, German, Greek and Italian. Meanwhile the theoretical work on syntax and meaning has continued, but there have been no breakthroughs in machine translation. The ambiguity pervading natural language continues to limit the possibilities, for reasons I shall examine more fully below.

I turn next to word processing, that is, to software that aids in the preparation, formatting and printing of text. Word processors deal only with the manipulation and display of strings of characters and hence only with superficial aspects of the structure of language. Even so, they pose technical problems quite central to the design of computer software. In some cases the end product of a word-processing program is no more than a sequence of lines of text. In others it is a complex layout of typographic elements, sometimes with drawings intercalated. In still others it is a structured document, with chapter headings, section numbers and so on, and with a table of contents and an index compiled by the program.

The key problems in designing word-processing software center on issues of representation and interaction. Representation is the task of devising data structures that can be manipulated conveniently by the software but still make provision for the things that concern the user of the system, say the layout of the printed page. Interaction takes up the issue of how the user expresses instructions and how the system responds.

Consider the fundamental problem of employing the data-storage devices of a computer to hold an encoded sequence of natural-language characters. The first devices that encoded text were card-punch and teletype machines, and so the earliest text-encoding schemes were tailored to those devices. The teletype machine is essentially a typewriter that converts key presses into numerical codes that can be transmitted electronically;



**MACHINE TRANSLATION** of text from one language into another was thought to be quite feasible in the 1950's, when the effort was undertaken. In the first step of the process (a) the computer would search a bilingual dictionary to find translations of the individual words in a source sentence (in this case Spanish equivalents of the words in the sentence "Did you see a white cow?"). Next the translated words would be rearranged according to the grammar of the target language (b). The changes at this stage could include excision or addition of words. Finally, the morphology of the translation (for example the endings of words) would be adjusted (c).

thus there are teletype codes for most of the keys on a typewriter. The codes include the alphabetic characters A through Z, the digits 0 through 9 and common punctuation marks such as the period and the comma. Standards are harder to establish, however, for symbols such as #, @, \$ and }. And what about keys that print nothing, such as the tab key, the carriage-return key and the backspace key?

The difficulties that arise in choosing standards can be illustrated by one peculiarity of text encoding. The teletype code distinguishes between a carriage return (which returns the type carriage to the beginning of the line without advancing the paper) and a line feed (which advances the paper without repositioning the carriage). Hence the end of a line is marked by a sequence of two characters: a carriage return and a line feed. One code would suffice, and so some programs eliminate either the carriage return or the line feed, or they replace both characters with another code entirely. The problem is that various programs employ different conventions, so that lines encoded by one program may not be readable by another.

The problems become worse when a full range of characters—punctuation marks, mathematical symbols, diacritical marks such as the umlaut—is considered. Moreover, word processing is now being extended to languages such as Chinese and Japanese, which require thousands of ideographic characters, and to languages such as Arabic and Hebrew, which are written from right to left. Coding schemes adequate for English are useless for alphabets with thousands of characters. It should be said that the schemes continue to vary because political and economic forces play a role in the design of computer systems. A given manufacturer wants to promulgate a standard that suits its own equipment; thus some present-day standards exist because they were offered by a vendor that dominates a market. On the other hand, technical matters such as the efficiency of certain software running on certain hardware perpetuate differences in detail. It will be quite a while before universal standards emerge and users gain the ability to transport text from one word-processing system to any other.

Encoding schemes aside, there is the form of the letters themselves. On a typewriter keyboard an A is simply an A. Typographically, however, an A is an A or an A or an A. In the new field of digital typography the computer is a tool for the design and presentation of forms of type. Some of the efforts in the field are applied to the forms themselves: in particular the representation of characters as composites of dots and spaces. Additional efforts go into the devising of code for the computer stor-

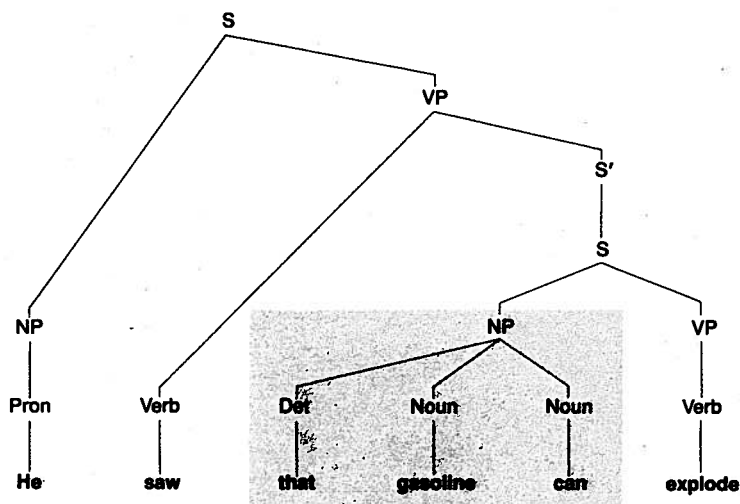
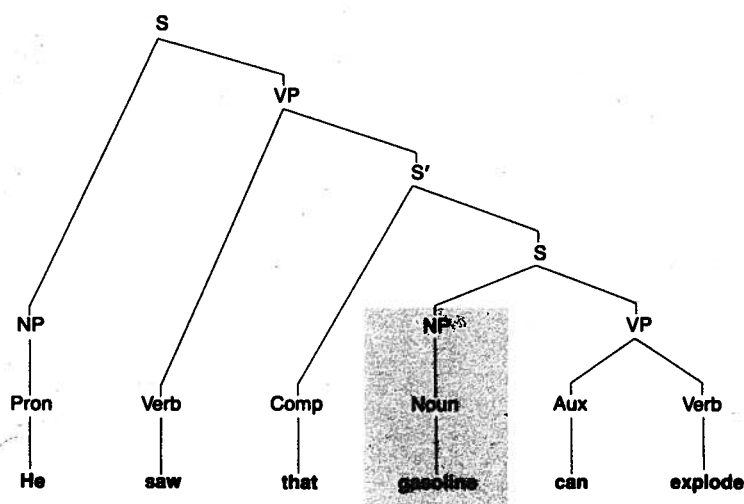
Stay away from the bank.

bank n 1. the rising terrain that borders a river or lake.

bank n 2. an establishment for the deposit, loan, issuance and transmission of money.

**AMBIGUOUS MEANINGS** permeate natural languages (that is, languages that people speak and write) and thus subvert the attempt to have computers translate text from one language into another. Here lexical ambiguity, the simplest type of ambiguity, is diagrammed. In lexical ambiguity a word in a sentence has more than one possible meaning. In this case the word is "bank" (color), which might equally well refer to either a river or a financial institution. A translator must choose. The following four illustrations show more complex types of ambiguity.

He saw that gasoline can explode.



**STRUCTURAL AMBIGUITY** arises when a sentence can be described by more than one grammatical structure. Here the conflicting possibilities for the sentence "He saw that gasoline can explode" are displayed in the form of grammatical "trees." In one of the trees the sentence has a subordinate clause whose subject is "gasoline" (color); the sentence refers to the recognition of a property of that substance. In the other tree "gasoline can" is part of a noun phrase (NP) meaning a container of gasoline; the sentence refers to the sight of a specific explosion.

age of text that combines different fonts (such as Times Roman and Helvetica) and different faces (such as *italic* and **boldface**).

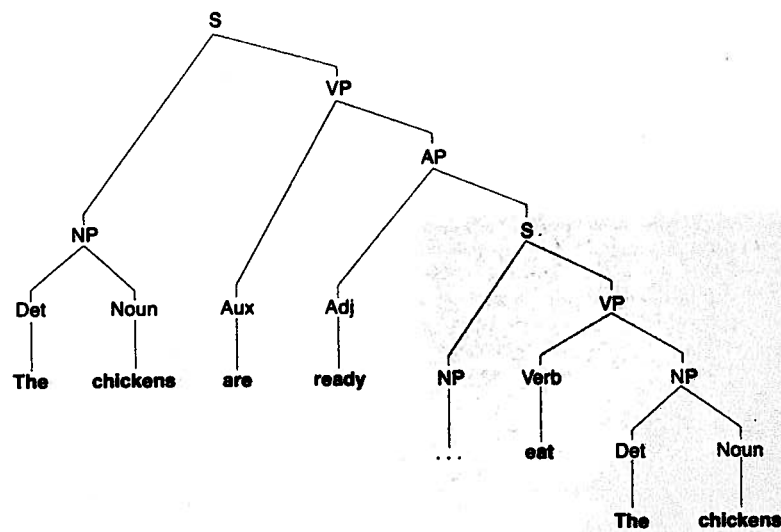
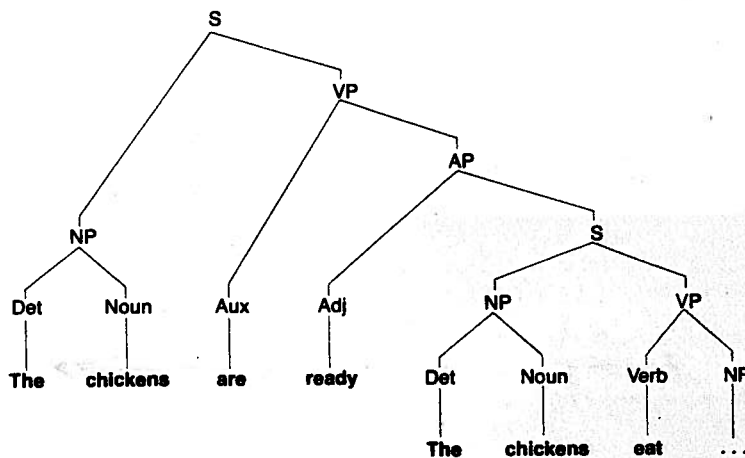
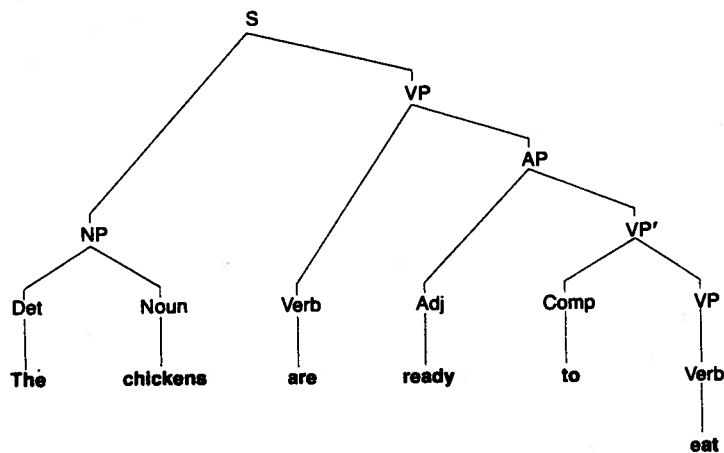
So far I have dealt only with stored sequences of characters. Yet one of the major tasks of a word-processing program is to deal with margins and spacing—with the “geography” of the printed page. In the typesetting language called TEX commands that specify non-standard characters, change the style of type, set the margins and so on are embedded in the text [see top illustration on page 138]. A command to TEX is distinguished from ordinary text by the backslash character (\). The stored text is “compiled” by the TEX program, which interprets the embedded commands in order to create a printed document in the specified format.

The compiling is quite complex, and a good deal of computation is often needed to get from code created by means of a word-processing program to code that readily drives a printer or a typesetting machine. An algorithm that justifies text (fills the full width of each line of type) must determine how many words will fit in a line, how much space should be inserted between the words and whether a line would be improved by dividing and hyphenating a word. The algorithm may also take actions to avoid visual defects such as a line with wide interword spacing followed by a line that is very compact. Positioning each line on the page is further complicated by the placement of headings, footnotes, illustrations, tables and so on. Mathematical formulas have their own typographic rules.

TEX and similar programs are primitive with respect to another aspect of word processing: the user interface. The high-resolution display screens becoming available are now making it possible for the computer to display to the user a fair approximation of the pages it will print, including the placement of each item and the typeface to be employed. This suggests that the user should not have to type special command sequences but might instead manipulate page geography directly on the screen by means of the computer keyboard and a pointing device such as a “mouse.” The resulting interface between the computer and the user would then fall into the class of interfaces known as WYSIWYG, which stands for “What you see is what you get.”

It is worth noting that programs for manipulating text are called different things by different professions. Programmers call them text editors, but in business and publishing they are referred to as word processors; in the latter fields an editor is a person who works to improve the quality of text. Computer software is emerging to aid in this

The chickens are ready to eat.



**DEEP-STRUCTURAL AMBIGUITY** arises when a sentence has a single apparent structure but nonetheless has more than one possible meaning. In this example the sentence is “The chickens are ready to eat.” Its grammatical structure (top) leaves the role of the chickens ambiguous: in one interpretation they will eat; in the other they will be eaten. Deep-structure trees make the chickens’ role explicit: they are the subject of the sentence (middle), in which case their food is undetermined, or they are the object (bottom), and their eaters are undetermined.

more substantive aspect of editing. It deals with neither the visual format of language nor the conceptual content but with spelling, grammar and style. It includes two kinds of programs: mechanized reference works and mechanized correctness aids.

An example of a mechanized reference work is a thesaurus program designed so that when the writer designates a word, a list of synonyms appears on the display screen. In advanced systems the thesaurus is fully integrated into the word-processing program. The writer positions a marker to indicate the word to be replaced. The thesaurus is then invoked; it displays the alternatives in a "window" on the screen. The writer positions the marker on one of the alternatives, which automatically replaces the rejected word.

The design of such a program involves both linguistic and computational issues. A linguistic issue is that the mechanism for looking up a word should be flexible enough to accept variant forms. For example, the store of information pertaining to "endow" should be accessible to queries about "endowed," "endowing," "endows" and even "unendowed" or "endowment." Recognizing the common root in such words calls for a morphological analysis, which can be done by techniques developed in the course of work on machine translation. Computational issues include devising methods for storing and searching through a thesaurus or a dictionary, which must be fairly large to be useful.

A correctness aid deals with spelling, grammar and even elements of style. The simplest such programs attempt to match each word in a text with an entry in a stored dictionary. Words that have no match are flagged as possible misspellings. Other programs look for common grammatical errors or stylistic infelicities. For example, the Writer's Workbench software developed at AT&T Bell Laboratories includes programs that search for repeated words, such as "the the" (a common typing mistake), for incorrect punctuation such as "?." and for wordy phrases such as "at this point in time." A different correctness aid calls attention to "pompous phrases" such as "exhibit a tendency" and "arrive at a decision" and suggests simpler replacements such as "tend" and "decide." Still another correctness aid searches for gender-specific terms such as "mailman" and "chairman" and suggests replacements such as "mail carrier" and "chairperson."

In addition to searching a text for particular strings of characters, some correctness-aid programs do statistical analyses. By calculating the average length of sentences, the length of words and similar quantities, they compute a "readability index." Passages that score poorly can be brought to the writer's attention. No program is yet able to make a comprehensive grammatical analysis of a text, but an experimental system called Epistle, developed at the International Business Machines Corporation, makes some grammatical judgments. It employs a grammar of

400 rules and a dictionary of 130,000 words. As with all software that tries to parse text without dealing with what the text means, there are many sentences that cannot be analyzed correctly.

Is there software that really deals with meaning—software that exhibits the kind of reasoning a person would use in carrying out linguistic tasks such as translating, summarizing or answering a question? Such software has been the goal of research projects in artificial intelligence since the mid-1960's, when the necessary computer hardware and programming techniques began to appear even as the impracticability of machine translation was becoming apparent. There are many applications in which the software would be useful. They include programs that accept natural-language commands, programs for information retrieval, programs that summarize text and programs that acquire language-based knowledge for expert systems.

No existing software deals with meaning over a significant subset of English; each experimental program is based on finding a simplified version of language and meaning and testing what can be done within its confines. Some investigators see no fundamental barrier to writing programs with a full understanding of natural language. Others argue that computerized understanding of language is impossible. In order to follow the arguments it is important to examine the basics of how a language-understanding program has to work.

A language-understanding program needs several components, corresponding to the various levels at which language is analyzed [see illustrations on pages 138-144]. Most programs deal with written language; hence the analysis of sound waves is bypassed and the first level of analysis is morphological. The program applies rules that decompose a word into its root, or basic form, and inflections such as the endings -s and -ing. The rules correspond in large part to the spelling rules children are taught in elementary school. Children learn, for example, that the root of "baking" is "bake," whereas the root of "barking" is "bark." An exception list handles words to which the rules do not apply, such as forms of the verb "be." Other rules associate inflections with "features" of words. For example, "am going" is a progressive verb: it signals an act in progress.

For each root that emerges from the morphological analysis a dictionary yields the set of lexical categories to which the root belongs. This is the second level of analysis carried out by the computer. Some roots (such as "the") have only one lexical category; others have several. "Dark" can be a noun or

David wants to marry a Norwegian.

$\exists x \text{ Norwegian}(x) \wedge \text{Want}(\text{David}, [\text{Marry}(\text{David}, x)])$

$\text{Want}(\text{David}, [\exists x \text{ Norwegian}(x) \wedge \text{Marry}(\text{David}, x)])$

**SEMANTIC AMBIGUITY** arises when a phrase can play different roles in the meaning of a sentence. Here the roles of the phrase "a Norwegian" become explicit when the sentence "David wants to marry a Norwegian" is "translated" into a logical form based on the notation called predicate calculus. According to one interpretation, the speaker of the sentence has a particular person in mind and has chosen nationality as a way to specify who. Hence the sentence means: There exists ( $\exists$ ) an  $x$  such that  $x$  is Norwegian and ( $\wedge$ )  $x$  is the person David wants to marry. According to another interpretation, neither David nor the speaker has any particular person in mind. David might be going to Norway hoping to meet someone marriageable.

She dropped the plate on the table and broke it.

She dropped the plate on the table and broke [the plate].

She dropped the plate on the table and broke [the table].

**PRAGMATIC AMBIGUITY** arises when a sentence is given more than one possible meaning by a word such as the pronoun "it." Suppose a computer is given the sentence shown in the illustration. If the computer has access to stored knowledge of the grammar of English sentences but lacks access to commonsense knowledge of the properties of tables and plates, the computer could infer with equal validity that the table was broken or that the plate was broken.

**a** \inset  
This is a sample of a {\italic justified} piece of text, which contains {\eightpoint small letters {\bold and }} {\bigFont big ones}. It includes foreign words such as {\quote pe\~ na\quote—which is Spanish—and foreign letters like {\alpha\ and {\aleph, which can be baffling, and includes one {\hskip 1.3in wide space.

**b**

01110100	01100101	01110010	01110011	00000000	00100111	00101101	11010011	00001000	01100001	01101110	01100100	00000000
t	e	r	s	NEW ENTITY	FONT CODE	X-POSITION	Y-POSITION	X-INCREMENT	a	n	d	NEW ENTITY

00110100	00110001	10110110	00101101	01100010	01101001	01100111	00100000	01101111	01101110	01100101	01101011	00101110
FONT CODE	X-POSITION	Y-POSITION	X-INCREMENT	b	i	g	SPACE	o	n	e	s	.

00000000	00000001	10101111	10110110	00101100	01001001	01110100	00100000	01101001	...			
NEW ENTITY	FONT CODE	X-POSITION	Y-POSITION	X-INCREMENT	i	t	SPACE	i				

**c** This is a sample of a *justified* piece of text, which contains small letters and **big ones**. It includes foreign words such as “peña”—which is Spanish—and foreign letters like α and №, which can be baffling, and includes one wide space.

**WORD PROCESSING**, that is, the computer-aided preparation and editing of text, requires several representations of the text, because the format best for interactions between the software and its user is not efficient for sending instructions to a printing machine, nor can it efficiently give a preview of the result of the printing. In the typesetting language TEX the user's typed input (a) includes commands that specify nonstandard characters, change the style of type, set margins

and so on. Such commands are distinguished by a backslash (\). The TEX software “compiles” the input, producing computer code that will drive a printing machine (b). To that end the code is divided into “entities,” each of which specifies the typeface and the starting position for a sequence of words. Coded “X increments” space out the words to fill the distance between margins on the printed page; thus they “justify” lines of type. The printed page (c) shows the result.

an adjective; “bloom” can be a noun or a verb. In some instances the morphological analysis limits the possibilities. (In its common usages “bloom” can be a noun or a verb, but “blooming” is only a verb.) The output of the morphological and lexical analysis is thus a sequence of the words in a sentence, with each word carrying a quantity of dictionary and feature information. This output serves in turn as the input to the third component of the program, the parser, or syntactic-analysis component, which applies rules of grammar to determine the structure of the sentence.

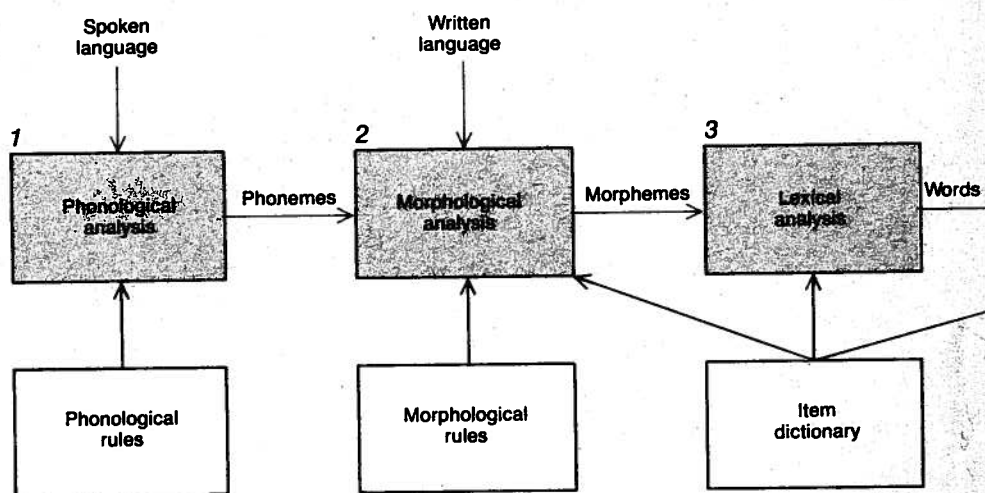
Two distinct problems arise in designing an adequate parser. The first problem is the specification of a precise set of rules—a grammar—that determines the set of possible sentence structures in a language. Over the past 30 years much work in theoretical linguistics has been directed toward devising formal linguistic systems: constructions in which the syntactic rules of a language are stated so precisely that a computer could employ them to analyze the language. The generative transformational grammars invented by Noam Chomsky of the Massachusetts Institute of Technology were the first comprehensive attempt; they specify the syntax of a language by means of a set of rules whose mechanical application generates all allowable structures.

The second problem is that of the parsing itself. It is not always possible to tell, when a part of a sentence is encoun-

tered, just what role it plays in the sentence or whether the words in it go together. Take the sentence “Roses will be blooming in the dark gardens we abandoned long ago.” The words “in the dark” might be interpreted as a complete phrase; after all, they are grammatically well formed and they make sense. But the phrase cannot form a coherent unit in a complete analysis of the sentence because it forces “Roses will be blooming in the dark” to be interpreted

as a sentence and therefore leaves “gardens we abandoned long ago” without a role to play.

Parsers adopt various strategies for exploring the multiple ways phrases can be put together. Some work from the top down, trying from the outset to find possible sentences; others work from the bottom up, trying local word combinations. Some backtrack to explore alternatives in depth if a given possibility fails; others use parallel processing



**COMPUTERIZED UNDERSTANDING OF LANGUAGE** requires the computer to draw on several types of stored data (white boxes) and perform several levels of analysis (colored boxes). If the language is spoken, the first analysis is phonological (1): the computer analyzes the sound waves. If the language is written, the first analysis is morphological (2): the computer decomposes each word into its root, or basic form, and inflections (for example -ing). Next is lexi-

to keep track of a number of alternatives simultaneously. Some make use of formalisms (such as transformational grammar) that were developed by linguists. Others make use of newer formalisms designed with computers in mind. The latter formalisms are better suited to the implementation of parsing procedures. For example, "augmented-transition networks" express the structure of sentences and phrases as an explicit sequence of "transitions" to be followed by a machine. "Lexical-function grammars" create a "functional structure" in which grammatical functions such as head, subject and object are explicitly tied to the words and phrases that serve those functions.

Although no formal grammar successfully deals with all the grammatical problems of any natural language, existing grammars and parsers can handle well over 90 percent of all sentences. This is not entirely to the good. A given sentence may have hundreds or even thousands of possible syntactic analyses. Most of them have no plausible meaning. People are not aware of considering and rejecting such possibilities, but parsing programs are swamped by meaningless alternatives.

The output of a parsing program becomes the input to the fourth component of a language-understanding program: a semantic analyzer, which translates the syntactic form of a sentence into a "logical" form. The point is to put the linguistic expressions into a form that makes it possible for the computer to apply reasoning procedures and draw inferences. Here again there are competing theories about what representation is most appropriate. As with parsing, the key issues are effectiveness and efficiency.

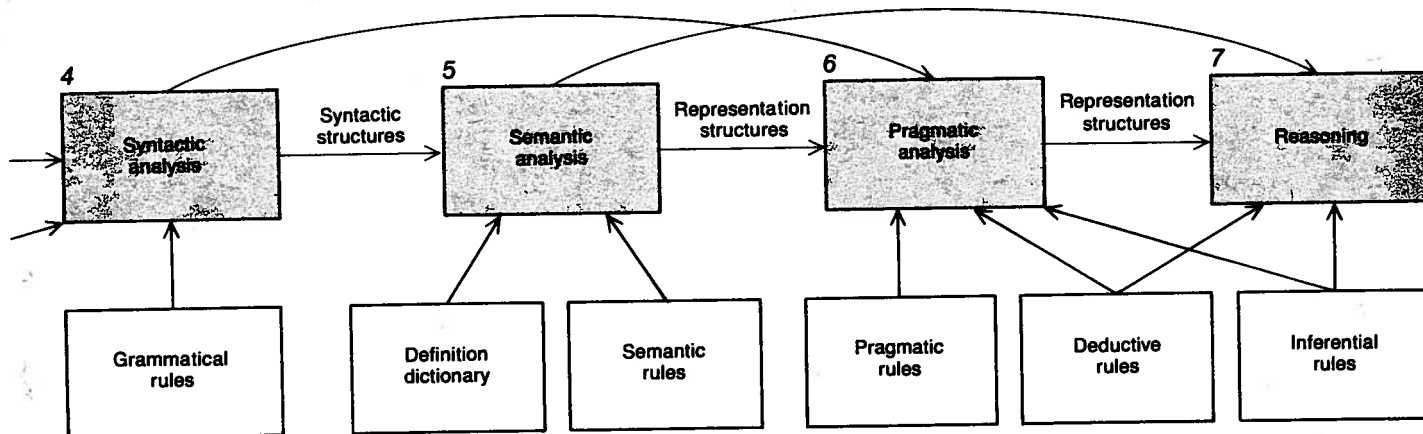
Effectiveness depends on finding the appropriate formal structures to encode the meaning of linguistic expressions. One possibility is predicate calculus, which employs the quantifiers  $\forall$  to mean "all" and  $\exists$  to mean "there exists." In predicate calculus "Roses will be blooming..." is equivalent to the assertion "There exists something that is a rose and that is blooming..." This entails a difficulty. Is one rose adequate to represent the meaning of "roses will be blooming," or would it be better to specify two or more? How can the computer decide? The dilemma is worsened if a sentence includes a mass noun such as "water" in "Water will be flowing..." One cannot itemize water at all. In designing a formal structure for the meaning of linguistic expressions many similar problems arise from the inherent vagueness of language.

Efficiency must also be considered, because the computer will employ the logical form of a sentence to draw inferences that in turn serve both the analysis of the meaning of the sentence and the formulation of a response to it. Some formalisms, such as predicate calculus, are not directly amenable to efficient computation, but other, more "procedural" representations have also been devised. Consider the effort to answer the question "Are there flowers in the gardens we abandoned long ago?" The computer needs to know that roses are flowers. This knowledge could be represented by a formula in predicate calculus amounting to the assertion "Everything that is a rose is a flower." The computer could then apply techniques developed for mechanical theorem-proving to make the needed deduction. A different approach would be to give certain inferences a privileged computational status. For example, basic clas-

sificational deductions could be represented directly in data structures [see bottom illustration on page 144]. Such deductions are required constantly for reasoning about the ordinary properties of objects. Other types of fact (for example that flowers need water in order to grow) could then be represented in a form closer to predicate calculus. The computer could draw on both to make inferences (for example that if roses do not get water, they will not grow).

A good deal of research has gone into the design of "representation languages" that provide for the effective and efficient encoding of meaning. The greatest difficulty lies in the nature of human commonsense reasoning. Most of what a person knows cannot be formulated in all-or-nothing logical rules; it lies instead in "normal expectations." If one asks, "Is there dirt in the garden?" the answer is almost certainly yes. The yes, however, cannot be a logical inference; some gardens are hydroponic, and the plants there grow in water. A person tends to rely on normal expectations without thinking of exceptions unless they are relevant. But little progress has been made toward formalizing the concept of "relevance" and the way it shapes the background of expectations brought to bear in the understanding of linguistic expressions.

The final stage of analysis in a language-understanding program is pragmatic analysis: the analysis of context. Every sentence is embedded in a setting: it comes from a particular speaker at a particular time and it refers, at least implicitly, to a particular body of understanding. Some of the embedding is straightforward: the pronoun "I" refers to the speaker; the adverb "now" refers to the moment at which the sen-



cal analysis (3), in which the computer assigns words to their lexical category (noun, for instance) and identifies "features" such as plurals. Then comes syntactic analysis, or parsing (4): the application of rules of grammar to yield the structure of the sentence. After that comes semantic analysis (5). Here the sentence is converted into a

form that makes it amenable to inference-drawing. The final stage is pragmatic (6): it makes explicit the context of the sentence, such as the relation between the time at which it is spoken and the time to which it refers. The computer is now in a position to draw inferences (7), perhaps in preparation for responding to the sentence.

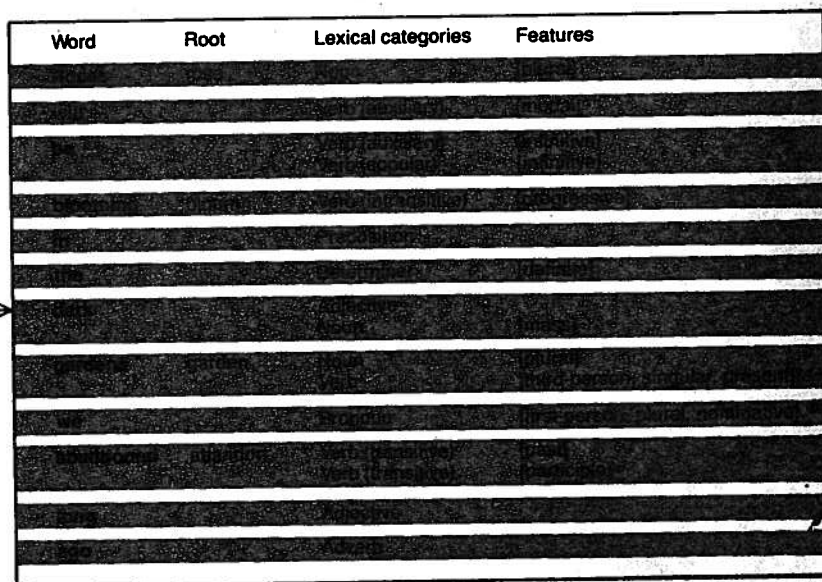
Still other types of embedding are not signaled by a troublesome word such as "we." The sentence "Roses will be blooming..." presupposes the identification of some future moment when the roses will indeed be in bloom. Thus the sentence might have followed the sentence "What will it be like when we get home?" or "Summer is fast upon us." Similarly, the noun phrase "the dark gardens we abandoned long ago" has a context-dependent meaning. There may be only one instance of gardens in which we have been together; there may be more than one. The sentence presupposes a body of knowledge from which the gardens are identifiable. The point is that a phrase beginning with "the" rarely specifies fully the object to which it refers.

events attending a meal in a restaurant. (A particular waiter or waitress serves any given customer.) In more complex cases an analysis of the speaker's goals and strategies can help. If one hears "My math exam is tomorrow, where's the book?" one can assume that the speaker intends to study and that "the book" means the mathematics text employed in a course the speaker is taking. The approach is hampered by the same difficulty that besets the representation of meaning: the difficulty of formalizing the commonsense background that determines which scripts, goals and strategies are relevant and how they interact. The programs written so far work only in highly artificial and limited realms, and it is not clear how far such programs can be extended.

**A**t first it might seem possible to distinguish "literal" uses of language from those that are more metaphorical or poetical. Computer programs faced with exclusively literal language could

The limitations on the formalization of contextual meaning make it impossible at present—and conceivably forever—to design computer programs that come close to full mimicry of human language understanding. The only programs in practical use today that attempt even limited understanding are natural-language “front ends” that enable the user of a program to request information by asking questions in English. The program responds with English sentences or with a display of data.

Some more recent front-end interfaces have been designed with practical applications in mind. A person wanting access to information stored in the computer types natural-language sentences



sentence, with their roots, their lexical categories and their features. "Blooming," for instance, is a progressive verb: it signifies an act in progress. The data serve as input for the syntactic level of analysis: the parsing of the sentence. Here the surface, or grammatical, structure of "Roses will be blooming..." is put in the form of a tree. Pro-

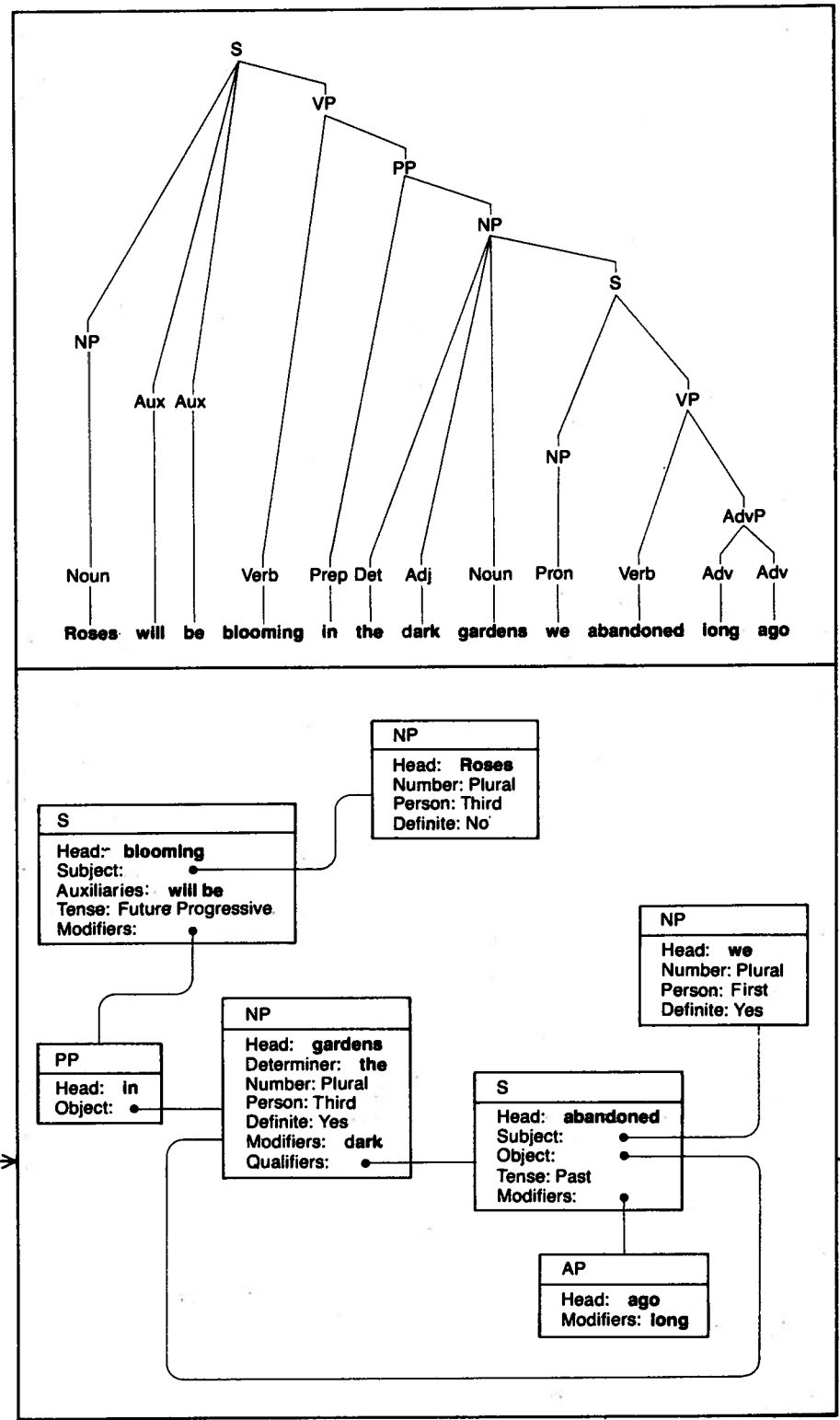
that the computer interprets as queries. The range of the questioning is circumscribed by the range of the data from which answers are formulated; in this way words can be given precise meaning. In a data base on automobiles, for example, "dark" can be defined as the colors "black" and "navy" and nothing more than that. The contextual meaning is there, but it is predetermined by the builder of the system, and the user is expected to learn it.

The main advantage of a natural-language front end is that it presents a low initial barrier to potential users. Someone invited to pose a question in English is usually willing to try, and if the computer proves unable to handle the specific form of the question, the user is probably willing to modify the wording until it works. Over time the user will learn the constraints imposed by the system. In contrast, a person who must learn a specialized language in order to formulate a question may well feel that an inordinate amount of work is being demanded.

I want finally to look at a rather new type of system called a coordinator. In brief it replaces standard electronic mail with a process that aids the generation of messages and monitors the progress of the resulting conversations. Coordinators are based on speech-act theory, which asserts that every utterance falls into one of a small number of categories. Some speech acts are statements: "It's raining." Some are expressive: "I'm sorry I stepped on your toe." Some are requests: "Please take her the package" or "What is your name?" Some are commitments: "I'll do it tomorrow." Some

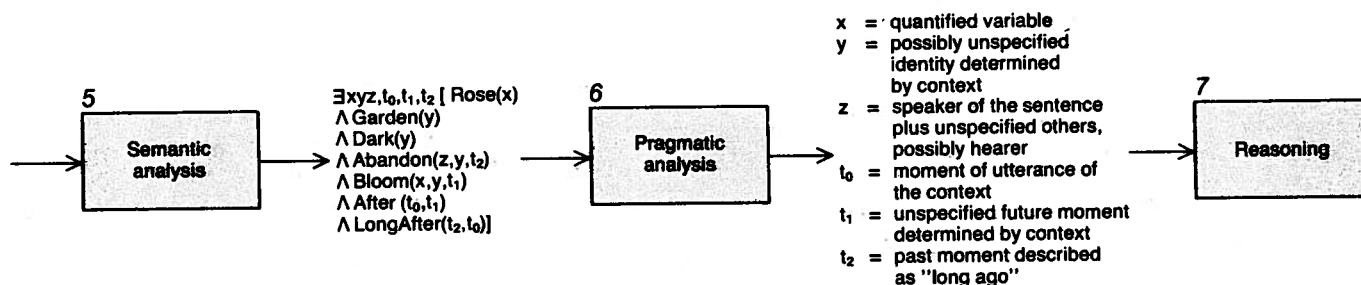
are declarative: "You're fired." (Declaratives differ from statements in that they take effect by virtue of having been said.) The classification of speech acts is useful because acts in the various categories do not occur at random. Each

speech act has "felicity conditions" under which it is an appropriate thing to say and "conditions of satisfaction" under which it is fulfilled. For example, a request or a commitment carries with it, either implicitly or explicitly, a time by which it should be satisfied. Moreover,



sumably the computer discards numerous incorrect trees. For example, it discards a tree in which "Roses will be blooming in the dark" is construed as a sentence. The deep structure of "Roses will be blooming..." is put in the form of a functional-structure diagram. There the relations between the parts of a sentence become explicit; they are

shown by strings between boxes. Some relations were explicit in the surface structure (for example that "roses" is the subject of "blooming"). Others were not (for example that "garden" is the object of "abandoned"). The syntactic analysis is supplied to the final stages of the program, which appear in the top illustration on the next page.



**ANALYSES CONCLUDE** with the conversion of the syntactic structure of "Roses will be blooming..." into a form from which the computer can draw inferences. In this example the conversion is based on predicate calculus; thus the semantic-analysis module of the hypothetical software represents the logical content of "Roses will be blooming..." by symbols that can be translated as "x is a rose and y is a garden and y is dark..." Finally, the pragmatic-analysis module

specifies what is known about the variables  $x, y, z, t_0, t_1$  and  $t_2$ . The variable  $x$ , for example, is "quantified": it declares the existence of something instead of identifying a particular object. In other words, the computer takes "roses" as referring to roses in general, not to particular roses. Hence roses is not a "definite" noun. (That decision was made in the course of semantic analysis.) On the other hand,  $z$  remains ambiguous because it stands for the ambiguous pronoun "we."

each speech act is part of a conversation that follows a regular pattern. The regularity is crucial for successful communication.

As with every aspect of language, the full understanding of any given speech act is always enmeshed in the unarticulated background expectations of the speaker and the hearer. The speech act "I'll be here tomorrow" might be a prediction or a promise, and "Do you play tennis?" might be a question or an invitation. In spoken conversation intonation and stress play a prominent part in establishing such meaning.

Coordinator systems deal with the speech acts embodied in messages by specifying what needs to be done and when. The system does not itself attempt to analyze the linguistic content of messages. Instead the word-processing software at the sender's end asks the sender to make explicit the speech-act content of each message. A person may write "I'll be happy to get you that report" in the message itself but must add (with a few special keystrokes) that the

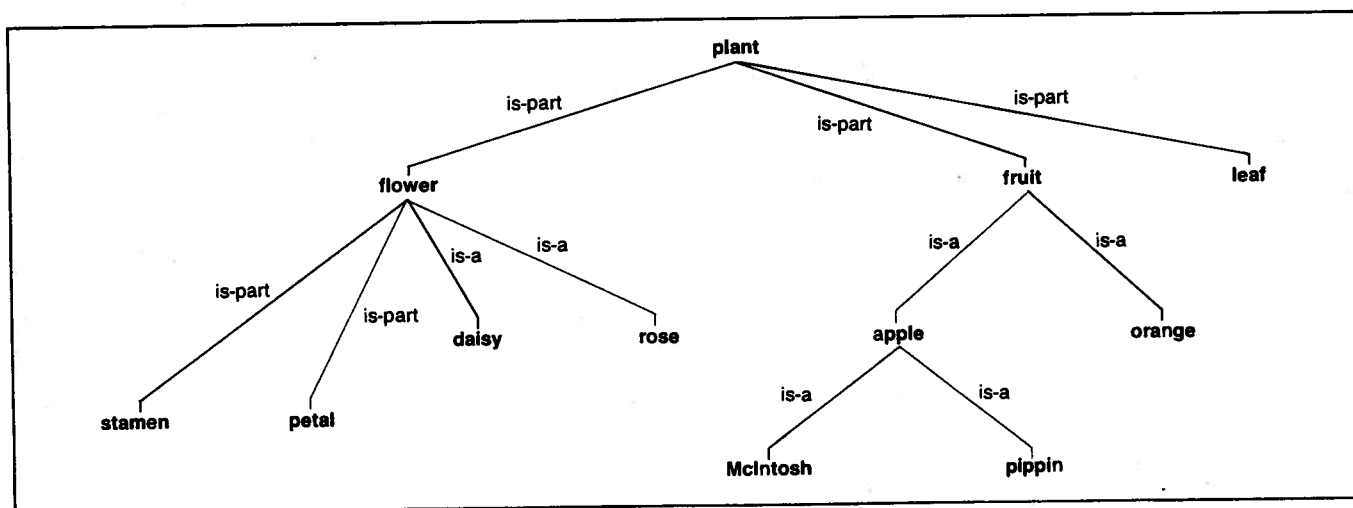
message is an ACCEPT of a particular REQUEST. The computer system can then keep track of messages and their interconnections. In particular the system can monitor the completion of conversations, calling the users' attention to cases in which something immediate is pending or in which an agreed-on time for satisfaction has not been met.

From a broad perspective, coordinators are just one member of a large family of software that gives users a structured medium in which language is augmented by explicit indications of how things fit together. Another type of software in this family provides tools for outlining and cross-indexing documents. Still another type is a computerized bulletin board that enables users to store and receive messages not addressed to a specific receiver. The messages are "posted" with additional structure that indicates their content and helps interested readers to find them.

The most obvious prediction about the future of computer software dealing with language is that the decreas-

ing cost of hardware will make applications that are possible but impractical today available quite widely in the future. Yet software that mimics the full human understanding of language is simply not in prospect. Some specific trends can be noted.

The first is that spoken language will get more emphasis. To be sure, the computerized understanding of spoken language presents all the difficulties of written language and more. Merely separating an utterance into its component words can vex a computer; thus hopes for a "voice typewriter" that types text from dictation are just as dim as hopes for high-quality machine translation and language-understanding. On the other hand, many useful devices do not require the analysis of connected speech. Existing systems that can identify a spoken word or phrase from a fixed vocabulary of a few hundred items will improve the interface between users and machines; the recent emergence of inexpensive integrated-circuit chips that



**SEMANTIC NETWORK** is a specialized form of stored data that represents logical relations so that certain types of inference can be drawn efficiently by a computer. Here a simple tracing of links in

the network (*color*) has yielded the inference that a pippin is a fruit and that a rose has petals. Facts not readily represented by a network can be represented in other ways, for example by predicate calculus.

process acoustic signals will facilitate the trend. Speech synthesizers that generate understandable utterances (although not in a natural-sounding voice) will also play an increasing role. Improved speech "compression" and encoding techniques will make acoustic messages and acoustic annotation of computer files commonplace.

A second trend in software dealing with language is that constraints on linguistic domain will be handled with increasing care and theoretical analysis. At several points in this article I have noted instances in which computers deal with meaning in an acceptable way because they operate in a limited domain of possible meanings. People using such software quickly recognize that the computer does not understand the full range of language, but the subset available is nonetheless a good basis for communication. Much of the commercial success of future software that deals with language will depend on the discovery of domains in which constraints on what sentences can mean still leave the user a broad range of language.

A third trend lies in the development of systems that combine the natural and the formal. Often it is taken for granted that natural language is the best way for people to communicate with computers. Plans for a "fifth generation" of intelligent computers are based on this proposition. It is not at all evident, however, that the proposition is valid. In some cases even the fullest understanding of natural language is not as expressive as a picture. And in many cases a partial understanding of natural language proves to be less usable than a well-designed formal interface. Consider the work with natural-language front ends. Here natural language promotes the initial acceptance of the system, but after that the users often move toward stylized forms of language they find they can employ with confidence, that is, without worrying about whether or not the machine will interpret their statements correctly.

The most successful current systems facilitate this transition. Some systems (including coordinators) mix the natural and the formal: the user is taught to recognize formal properties of utterances and include them explicitly in messages. Thus the computer handles formal structures, while people handle tasks in which context is important and precise rules cannot be applied. Other systems incorporate a highly-structured query system, so that as the user gains experience the artificial forms are seen to save time and trouble. In each case the computer is not assigned the difficult and open-ended tasks of linguistic analysis; it serves instead as a structured linguistic medium. That is perhaps the most useful way the computer will deal with natural language.



Chivas Regal • 12 Years Old Worldwide • Blended Scotch Whisky • 86 Proof  
© 1984 General Wine & Spirits Co., N.Y.