Natural language processing: Some recent trends

Aravind K. Joshi

Department of Computer and Information Science, University of Pennsylvania, Philadelphia, PA 19104, USA

In this article, I describe a few aspects of natural language processing (NLP), specifically some recent trends. I have chosen these topics as they cover a wide range of current efforts in NLP. They are also sufficiently diverse ranging from new avenues of research in grammars and parsing, statistical approaches to NLP, a relatively new trend in NLP, and integration of language and graphic/animation modalities, a new domain for investigating semantic and pragmatic aspects of language.

LANGUAGE (spoken and written) is central to all aspects of our communication. Therefore natural language processing systems (NLP), both current and future, are bound to play a crucial role in our communication with machines and even among ourselves. NLP systems include systems for speech recognition, language understanding and language generation. Spoken language systems are those that integrate speech and language systems. Such systems will provide, and to some extent already do so, an interface to databases and knowledge bases, for example, an airline information and reservation system, expert systems for scheduling, planning, and maintenance, among others. Text processing and message understanding systems are useful for extracting information from texts and formatting it in a variety of ways for further use. Language communication often occurs in two or more languages. Multilingual NLP has applications to a variety of multilingual tasks such as providing aids for translating foreign language correspondence, translating equipment manuals, and speech-to-speech translation in limited domains, among others. Finally, natural language in conjunction with graphic/animation modality provides very useful cooperative interfaces, especially in the instructional domains.

NLP is concerned with (i) the study of mathematical and computational models of the structure and function of language, its use, and its acquisition and (ii) the design, development, and implementation of a wide range of systems as mentioned above. On the theoretical side, the study involves mathematical and computational modeling of syntax, semantics, pragmatics (that is, certain aspects of the relationship of the speaker and the hearer, or user and the system in the case of an NLP system), and discourse aspects of language. These investigations are interdisciplinary and involve concepts in computer science including artificial intelligence, linguistics, logic, and psychology.

I cannot, obviously, survey the entire field of NLP in this limited space. I will not even provide a comprehensive survey of the three selected areas. My goal is to provide a short introduction (based on examples) to these three topics and justify the significance of the issues involved. I have provided the major references to these three topics, as well as to some other key topics not discussed here at all.

Many major topics have been omitted, all of which are very important to NLP. I have not discussed speech recognition and synthesis at all, and in the language area, I have not discussed many aspects of discourse structure, which are crucial to natural language understanding and generation and their applications to cooperative interfaces[1]. I have also not discussed the very important area of machine translation.

Grammars and parsers

Language has a hierarchical structure at various levels, in particular at the sentence level, which is the level we will be concerned with in this section. Almost every NLP system has a grammar and an associated parser. A grammar is a finite specification of a potentially infinite number of sentences, and a parser for the grammar is an algorithm that analyses a sentence and assigns one or more structural descriptions to the sentence according to the grammar, if the sentence can be characterized by the grammar. A structural description is a record of the derivational history of the sentence according to the grammar. The structural descriptions are necessary for further processing, for example, for semantic interpretation. Chomsky's[2] work on formal grammars in the late fifties was the beginning of the investigations of mathematical and computational modeling of grammars. He introduced a hierarchy of grammars (finite state grammars, context-free grammars, context-sensitive grammars, and unrestricted rewriting systems) and investigated their linguistic adequacy.

Many NLP systems are based on context-free

grammers (CFGs). We will briefly describe CFGs. A CFG, G, consists of a finite set of non-terminals (for example, S: sentence; NP: noun phrase; V: verb; ADV: adverb), a finite set of terminals (for example, *Harry*, *peanuts*, *likes*, *passionately*), and a finite set of rewrite rules of the form $A \rightarrow W$, where A is a non-terminal and W is a string of zero or more non-terminals and terminals. S is a special non-terminal called the start symbol. In Figure 1 we have a simple example of a CFG. The rewrite rules in the left column are called syntactic rules and the rules in the right column are called lexical rules, as these rules rewrite a non-terminal into terminals or lexical items. A derivation in a grammar begins with S, the start symbol. S is rewritten as a string of non-terminals and terminals, using a rewrite rule applicable to S. The new non-terminals are then rewritten according to the rewrite rules applicable to them, until no further rules can be applied. It is easy to see that the sentence *Harry likes peanuts passionately* can be generated by the grammar. In Figure 1, the tree shows the structural description assigned by the grammar to the sentence spelled out by the lexical items appearing at the frontier nodes of the tree. Here the derivation starts with the start symbol S. This symbol is then rewritten as the string NP VP. These two symbols are now rewritten (in any order) as the strings *Harry* and VP ADV respectively. The symbol VP is rewritten as the string V NP and ADV is rewritten as *passionately*. Finally, V is rewritten as *likes* and NP is rewritten as *peanuts*. The tree in Figure 1 is the result of these rewritings.

A Context-Free Grammar (CFG)

Syntactic Rules	*Lexical Rules*
S → NP VP	NP → **Harry**
VP → VP ADV	NP → **peanuts**
VP → V NP	V → **likes**
	ADV → **passionately**

Structural description assigned to the sentence:
Harry likes peanuts passionately

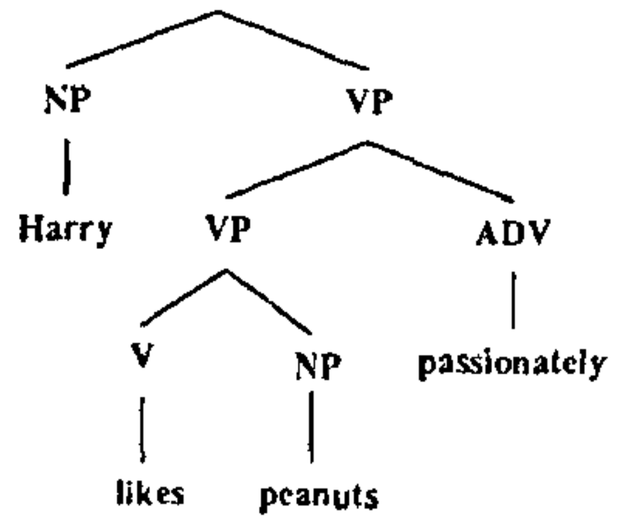


Figure 1. A context-free grammar.

A finite-state grammar is like a CFG, except that the rewrite rules are of the form $A \rightarrow aB$ or $A \rightarrow a$, where A and B are non-terminals and a is a terminal symbol. Finite-state grammars have been shown to be inadequate for modeling natural language structure. This is because there are dependencies that hold at unbounded distances. Some examples are given below (see for example the filler-gap dependencies described in the section on Mildly Context-Sensitive Grammars; see also the section on Statistical Approaches to Natural Language). A context-sensitive grammar is also like a CFG, except that the rewriting of a non-terminal is dependent on the context surrounding the non-terminal, unlike the rewrite rules in CFG where the rewriting is context-independent. Context-sensitive grammars appear to be adequate for describing natural language structures. However, the entire class of context-sensitive grammars appears to be too powerful in the sense that it is not constrained enough to characterize just the structures that arise in natural language.

CFGs, as defined above, are inadequate for a variety of reasons and need to be augmented. The two main reasons are as follows: (i) The information associated with a phrase (a string of terminals) is not just the atomic symbols used as non-terminals. A complex bundle of information (sets of attribute-value pairs, called feature structures) has to be associated with strings, the syntactic category of the phrase being only one such feature, for example. Appropriate structures and operations for combining them are needed together with a CFG skeleton; (ii) The string combining operation in a CFG is concatenation, that is, if u and v are strings, v concatenated with u gives the string $w = uv$, that is, u followed by v. More complex string combining as well as tree combining operations are needed to describe various linguistic phenomena. I will illustrate these two kinds of augmentations by some simple examples.

CFG-based unification grammars

A feature structure consists of a set of attribute–value pairs, where a value may be atomic or may be another feature structure. In Figure 2, the feature structure X_1 consists of a feature *cat* (*category*) whose value is NP and a feature *head* whose value is another feature structure. This feature structure has only one attribute, *agreement*, whose value is another feature structure with attributes *number* and *person* with values *singular* and *third* respectively. X_1 is a feature structure that can be

$$X_1 : \begin{bmatrix} \text{cat} : & \text{NP} \\ \text{head} : & \begin{bmatrix} \text{agreement} : \begin{bmatrix} \text{number} : & \text{singular} \\ \text{person} : & \text{third} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$X_2 : \begin{bmatrix} \text{cat} : & VP \\ \text{head} : & \begin{bmatrix} \text{form} : & \text{finite} \\ \text{subject} : & \begin{bmatrix} \text{agreement} : \begin{bmatrix} \text{number} : & \text{singular} \\ \text{person} : & \text{third} \end{bmatrix} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$X_0 \rightarrow X_1 \; X_2$$

$$X_0 : \begin{bmatrix} \text{cat} : & \text{S} \\ \text{head} : & \text{(value is the same as the value of the head feature of } X_2\text{)} \end{bmatrix}$$

Figure 2. CFG-based unification grammar.

appropriately associated with the phrase *Fido* because *Fido* is a noun-phrase NP whose number is singular and it is a third person noun-phrase. Similarly X_2 is a feature structure that can be appropriately associated with the phrase *snores* because *snores* is a verb with a tense (present), i.e. it is a finite verb and requires a subject that is singular and in the third person. The context-free rewriting rule $X_0 \rightarrow X_1 X_2$ can be interpreted as an instruction for combining the strings *Fido* and *snores* to give the string *Fido snores* and building the feature X_0 to be associated with it, as shown in Figure 2. This little example illustrates the main idea behind CFG-based unification grammars[3].

The main operation for combining feature structures is called unification. Given two feature structures A and B, we get a new feature structure C by unifying A and B, which has all the information in A and all the information in B and no more. Of course, if A and B have contradictory information, then A and B will fail to unify. In a CFG-based unification grammar, the CFG (context-free grammar) serves as a skeleton which defines the string combining operations. The objects that the grammar manipulates are feature structures. The feature structures are combined by the operation of unification as explained above. Thus in this type of unification grammar the grammar builds the string and the unifications of the appropriate feature structures (beginning with the feature structures associated with the lexical items, i.e. the words) build a feature structure associated with the string built by the grammar.

A variety of grammars such as generalized phrase structure grammar (GPSG)[4], head driven phrase structure grammar (HPSG)[5] and lexical functional grammar (LFG)[6] are essentially based on CFG-based unification grammars. An introduction to unification-based grammars appears in ref. 3. Unification is a very powerful operation and, unless restricted, CFG-based unification grammars are Turing Machine equivalent, that is, their computing power equals the power of a general-purpose computing machine with unlimited working tape. From a linguistic point of view, these grammars have to be restricted so that their descriptive power is no more than necessary, and from a computational point of view, they have to be restricted in order to yield efficient parsing algorithms[7]. Both these considerations form the basis for continued research in this area.

Mildly context-sensitive grammars

In any mathematical or computational grammar, a wide range of dependencies among the different elements in the grammar have to be described. Some examples of these dependencies are as follows: (i) Agreement features such as person, number, and gender. For example, in English, the verb agrees with the subject in person and number; (ii) Verb subcategorization, in which each verb specifies one (or more) subcategorization frames for their complements. For instance, *sleep* does not require any complement, (as in *Harry sleeps*), *like* requires one complement (as in *Harry likes peanuts*), *give* requires two complements (as in *Harry gives Susan a flower*), and so forth; (iii) Sometimes the dependent elements do not appear in their normal positions. In

Who$_i$ *did John invite* e_i

where e_i is a stand-in for *who*$_i$, *who*$_i$ is the filler for the gap e_i. The filler and the gap need not be at a fixed distance. Thus in *who*$_i$ *did Bill ask John to invite* e_i, the filler and the gap are more distant than in the previous sentence; (iv) Sometimes the dependencies are nested. In German, for example, one could have

Hans$_i$ *Peter*$_j$ *Marie*$_k$ *schwimmen*$_k$ *lassen*$_j$ *sah*$_i$
(Hans saw Peter make Marie swim)

where the nouns (arguments) and verbs are in nested order, as the subscripts indicate; (v) However, in Dutch, these dependencies are crossed, as for example, in

Jan$_i$ *Piet*$_j$ *Marie*$_k$ *zag*$_i$ *laten*$_j$ *zwemmen*$_k$
(Jan saw Piet make Marie swim).

There are, of course, situations where the dependencies have more complex patterns. Precise statements of such dependencies and the domains over which they operate constitute the major activity in the specification of a grammar. Mathematical and computational modeling

of these dependencies is one of the key areas in natural language processing. Many of these dependencies (for example, the crossed dependencies discussed above) cannot be described by context-free grammars[8–10]. This is easily seen from the well-known fact that CFGs are equivalent to the so-called push-down automata (PDAs) which have the storage discipline—last in first out. PDAs therefore can characterize nested dependencies.

In the context-free grammar (CFG) in Figure 1 the dependency between a verb (*likes*) and its two arguments [subject (NP) and object (NP)], is specified by means of two rules of the grammar. It is not possible to specify this dependency in a single rule without giving up the VP (verb phrase) node in the structure. That is, if we introduce a rule, S→NP V NP, then we can express the dependency in one rule, but then we cannot have VP in our grammar. Hence, if we regard each rule of a CFG as specifying the domain of locality, then the domain of locality for a CFG cannot locally (that is, in one rule) encode the dependency between a verb and its arguments, and still keep the VP node in the grammar.

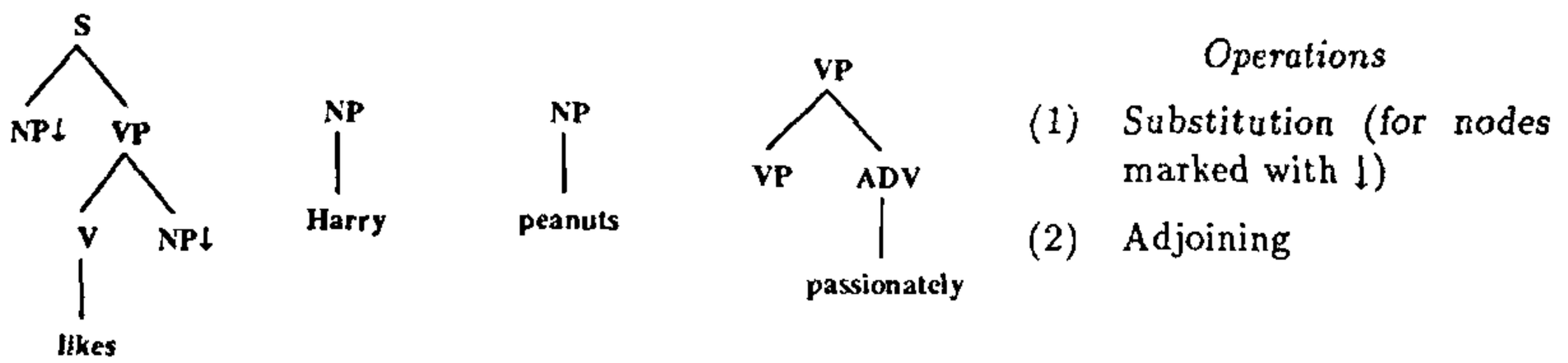
We will now describe briefly two grammars whose domain of locality is larger than that of a CFG.

In the tree-adjoining grammar (TAG) in Figure 3*a*, each word is associated with a structure (tree) (the word serves as an *anchor* for the tree) which encodes the dependencies between this word and its arguments (and therefore indirectly its dependency on other words which are anchors for structures that will fill up the slots of the arguments). Thus for *likes*, the associated tree encodes the arguments of *likes* (that is, the two NP nodes in the tree for *likes*) and also provides slots in the structure where they would fit. The trees for *Harry* and *peanuts* can be substituted respectively in the sujbect and object slots of the tree for *likes*. The tree for *passionately* can be inserted (adjoined) into the tree for *likes* at the VP node. The derivation in a TAG grammar is quite different from the derivation in a CFG. The tree in Figure 3*a* is a derived tree in the TAG shown in the figure. It is not the derivation tree. The derivation tree (for the derived tree shown in Figure 3*a*) will be a record of the history of the various adjoinings and substitutions carried to produce the tree in Figure 3*a*. This derivation tree is not shown in Figure 3*a*. In a TAG, the entire grammar consists of lexical items and their associated structures. There are universal operations, substitution and adjoining which describe how structures can be combined[11–13].

In the combinatory categorical grammar (CCG) in Figure 3*b*, each word is assigned a category, atomic or composite. The category for *Harry* and *peanuts* is NP, an atomic category. For *likes*, the category is (S\NP)/NP. This expression encodes the information that *likes* has two arguments. The category can be interpreted as a function, which when applied to an argument NP (the object) on the right, returns (S\NP), which is also a function. This function, when applied in

a. Tree Adjoining Grammar (TAG)

Lexical Trees



Operations

(1) Substitution (for nodes marked with ↓)

(2) Adjoining

b. Combinatory Categorial Grammars (CCG)

Lexical Categories

likes: (S\NP)/NP (composite)

Harry: NP (atomic), S/(S\NP) (composite)

peanuts: NP (atomic)

passionately: (S\NP)\(S\NP) (composite)

Operations

(1) function application

(2) function composition

Figure 3. Two grammar formalisms with domains of locality larger than the domain of locality for CFG.

turn to an argument NP (the subject) on the left, returns S (sentence). In this representation, (S\NP) serves the same role as VP. In a CCG, the entire grammar consists of lexical items and their category assignments. There are two universal operations, function application and function composition, which describe how categories are combined. Note that *passionately* is combined with *likes peanuts* by function composition. CCG also allows type raising. For example, *Harry* has the category NP, but we can also assign another category to *Harry*, namely S/(S\NP), that is, a function requiring a verb-phrase on the right and returning S. This category assignment is appropriate only if *Harry* is in the subject position[14,15]. Derivation in a CCG is the history of how a string is built by the successive use of the function application and composition operations. This history can be represented as a tree, not shown in Figure 3*b*. A CCG does not necessarily assign a unique phrase structure. The structure depends on the operations used and the order in which they were used. Different choices of operations and different orders of use will result in different phrase structure descriptions, even for unambiguous sentences.

Both CCG and TAG have domains of locality that are larger than that for CFG, because in each case all the arguments of the verb *likes* are encoded in structures associated with the verb and yet, the node VP(=S\NP in CCG) is available. The larger domain of locality allows TAG to completely factor out recursion from the domain of dependencies, thus localizing all dependencies in the elementary trees[12]. For the linguistic significance of CCG and TAG, see refs. 14,15,16–18.

TAG and CCG are very similar. In fact, they have been shown to be formally equivalent (with respect to their weak generative capacity, that is, the sets of sentences they generate). They are more powerful than CFG and belong to a class of grammars that we call mildly context-sensitive grammars (MCSG)[19]. This class preserves many of the essential properties of CFG and yet is able to provide enough power to capture a wide range of dependencies of language structure, such as the crossed dependencies we discussed earlier. Several other recent formalisms, for example, Linear Indexed Grammar and Head Grammar, have also been shown to be equivalent to TAGs[19–21]. This equivalence of a number of linguistically motivated grammars based on quite distinct insights into the structure of language has led to the search for invariances across this class of grammars, these invariances being more important in some sense than the individual grammars[19]. The study of mildly context-sensitive grammars and the study of their equivalences is one of the most active areas of investigation in mathematical linguistics during the last decade.

We have been implicitly assuming that a grammar assigns a unique structure to a sentence (assuming that the sentence is unambiguous). Thus for example *Harry likes peanuts* will be bracketed as follows (ignoring the phrase labels and ignoring some brackets not essential for our present purpose):

(a) (Harry (likes peanuts))

It is possible in a CCG to assign multiple structures to unambiguous sentences[14], as we have pointed out above. Thus CCG assigns the following two groupings to *Harry likes peanuts*:

(b) (Harry (likes peanuts))

(c) ((Harry likes) peanuts)

The justification for such multiple structures is their use in coordinations (for example, with *and*) and in defining intonational phrases. Thus the bracketing (b) is necessary for (d) and the bracketing (c) for (e).

(d) (Harry ((likes peanuts) and (hates cashews)))

(e) (((Harry likes) and (Bill hates)) cashews)

Also, (b) corresponds to the intonational phrasing if the previous context is (f) and (c) if the previous context is (g).

(f) Who likes peanuts? (Harry (likes peanuts))

(g) What does Harry like? ((Harry likes) peanuts)

The flexibility in the assignment of structure is achieved by giving up the notion of a canonical structure. Thus in Figure 3*b*, if *Harry* is assigned the category S/(S\NP), it can either combine with *likes* by function composition giving the structure in (c) above, or it can apply to the predicate *likes peanuts* to yield (b) above[14]. However, it is not necessary to give up the notion of canonical structure. It is possible to maintain a fixed structure at a certain level (at the level of elementary trees in a TAG, for example) and still achieve the kind of flexibility needed for examples shown above[13]. Similar results can be obtained in the Head-Driven Phrase Structure Grammar (HPSG) framework.

Parsing complexity

A parser for a grammar is an algorithm that assigns to a sentence one or more structural descriptions according to the grammar, if the sentence is generable by the grammar. Parsing of sentences according to different grammars and the complexity of this process

are important research areas in NLP. For a CFG a number of parsing algorithms are known and the time required to parse a sentence of length n is at most Kn^3, where K depends on the size of the grammar. This result extends to almost all CFG-based grammars used in NLP. The constant K can become very large however. In practice, of course, the worst case complexity is really not the important measure. Most parsers perform much better than the worst case on typical sentences. There are no mathematical results, as yet, to characterize the behaviour on typical sentences. Grammars that are more powerful than CFG are, of course, harder to parse, as far as the worst case is concerned. The grammars in the class of Mildly Context-Sensitive Grammars discussed earlier can all be parsed in polynomial time just as CFG, however, the exponent for n is 6 instead of 3.

A crucial problem in parsing is not just to get all possible parses for a sentence but to rank the parses according to some criteria. If a grammar is combined with statistical information (see next section), then that information can be used to provide this ranking. This is exactly what is done in many spoken language systems, that is systems that integrate speech recognition and language processing[22].

In our discussion so far, we have been assuming that the parser only handles complete sentences and the parser either succeeds in finding the parse(s) for a sentence or it fails. In practice, we want the parser to be flexible—that is, it should be able to handle fragments of sentences—and it should fail gracefully—that is, it should provide as much analysis as possible for as many fragments of the sentence as possible, even if it cannot glue all the pieces together. A parser with such properties based on the idea of deterministic parsing[23] has been described in ref. 24 and used in the construction of a large corpus of parsed text, a tree bank[25].

Finally, the actual grammars in major NLP systems are large, but even with this large size their coverage is not adequate. Building the grammar by hand soon reaches its limit and there is no guarantee that it will be increasingly better in coping with free text (say, text from a newspaper) by continuing to build it manually. Increasing attention is being paid now to automatically acquiring grammars from a large corpus[25]. See the following section for further details.

Statistical approaches to natural language processing

There is a long history of modeling language statistically. After all, some words occur more frequently than other words (for example, *the* occurs more frequently than *man*, which occurs more frequently than *aardvark*), some two-word sequences appear more frequently than some other two-word sequences (for example, *a man* occurs more frequently than *old man*, which occurs more frequently than *green man*), and so forth. Hence, it is reasonable to believe that language can be modeled statistically. A specific proposal along these lines was made by Shannon[26] in 1948. He viewed the generation process as modeled by stochastic processes, in particular, a Markov process. For our present purpose, we will characterize sentence generation by a finite state machine (Figure 4). Given a state diagram, we generate a sentence by starting with the initial state and then traversing the diagram from state to state and emitting the word labeling the arc between a pair of states. The process ends when we reach the final state. A probability is assigned to each state transition together with the emitted symbol, that is to a triple (S_i, a_j, S_k) representing the transition from state S_i to state S_k emitting the symbol a_j. Although such machines are clearly relevant to modeling language statistically, Chomsky[27] rejected the finite state machine characterization as inappropriate for modeling grammars, for

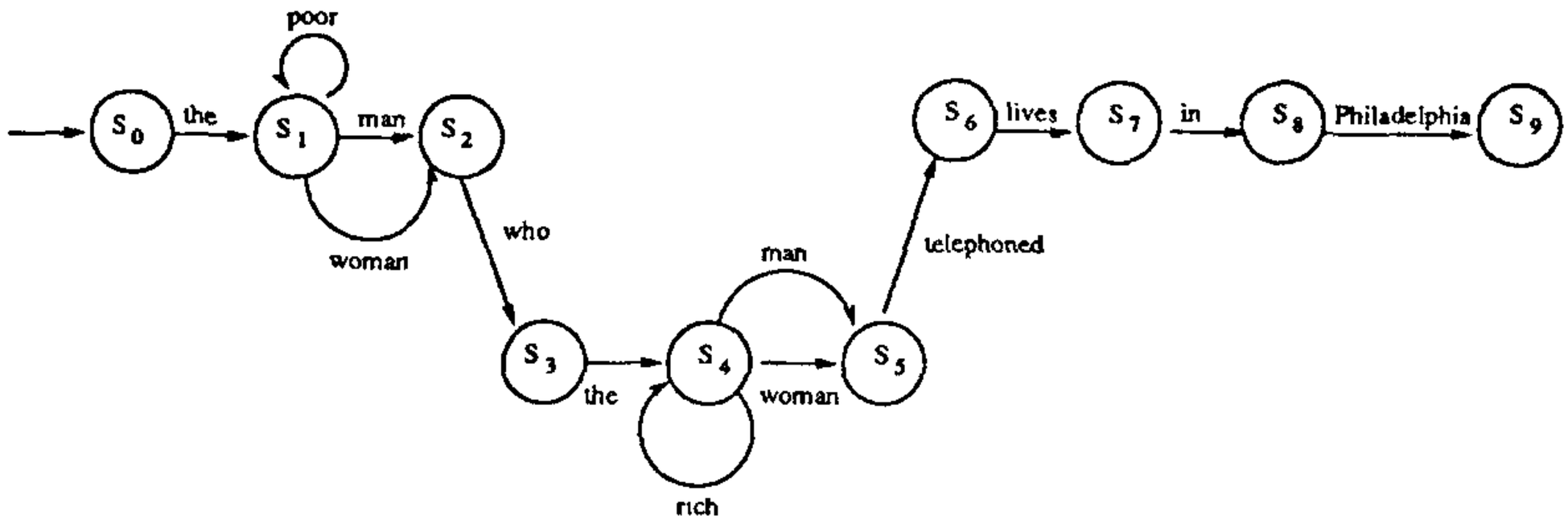


Figure 4. A finite state machine generating sentences.

the following reason: In Figure 4, *lives* is four words away from *man*, assuming that we did not follow the loop at S_4. Hence the dependency between these two words can be captured by the state sequence from S_2 to S_6. However, in the sentence *The man who the woman Harry met yesterday telephoned lives in Philadelphia*, (one that is a bit difficult to process but grammatical, and not generable by the machine in Figure 4), *lives* is now seven words away from *man*. Since more clauses can be embedded and each clause can be lengthened by adding adjectives or adverbs, the distance between *lives* and *man* can be made arbitrarily large and thus the number of states required to model language cannot be bounded. Hence a finite state machine is inadequate. Chomsky also rejected the possibility of associating the probability of a sentence with its grammaticality (the higher the probability, the higher the grammaticality of the sentence). This is because if we order the sequence of a given length (there will be W^n such sequences, if W is the number of words and n is the length of the sequences) according to the probabilities of the sequences then it will not be possible to sort out grammatical and ungrammatical sequences on the basis of this ranking[27]. Chomsky then developed structural models, such as the phrase structure grammar and transformational grammar, which formed the basis for almost all of the work in mathematical and computational linguistics up until the present.

Although Chomsky rejected the statistical models, he commented[27] 'Given the grammar of language, one can study the use of the language statistically in various ways; and the development of probabilistic models for the use of language (as distinct from the syntactic structure of language) can be rewarding.... One might seek to develop a more elaborate relation between statistical and syntactic structure than the simple order of approximation model we have rejected. I would certainly not care to argue that any such relation is unthinkable, but I know of no suggestion to this effect that does not have obvious flaws. ...'

Harris[28] around 1957, proposed a transformational theory motivated by the considerations of normalizing sentence structures (for the purpose of discourse analysis) so that the relevant co-occurrences among words can be stated in a local manner. Very roughly speaking, under this view, *The man who Harry met yesterday lives in Philadelphia*, is made up of **S1**: *The man lives in Philadelphia* and **S2**: *who Harry met* (which is a transformed version of **S3**: *Harry met the man*, with **S1** and **S3** sharing *the man*) and so on. There are clearly 'meaningful' statistical dependencies between *lives* and the subject noun *man* and the object of *in*, namely, *Philadelphia*, and between *met* and *Harry*, the subject of *met*, and *man* the object of *met*, but not 'meaningful' statistical dependencies between *lives* and *yesterday* or *met yesterday* (the one-word and two-word sequences before *lives*) and so on.

Although statistical approaches did not play a significant role in mathematical or computational linguistics, it is clear that the idea of somehow combining structural and statistical information was already suggested as early as the late fifties. Now in the nineties, we see a resurgence of these early ideas. There are two key reasons for this renewed interest. First, we now have some formal frameworks which appear to be suitable for combining structural and statistical information in a principled manner and second, there is now the possibility of using very large corpora, annotated in various ways, that can be used for reliably estimating the various statistics needed to deduce linguistic structure[25].

Hidden Markov Models (HMM) have played a crucial role in speech recognition. HMMs are derived from the theory of probabilistic functions of finite state Markov chains[29,30]. HMMs were introduced in the speech recognition domain in the early eighties and became very popular in the late eighties. They have also found use in the spoken language systems, i.e. systems that integrate speech and natural language. As we have already pointed out finite state models are not adequate for modeling the structure of natural language; more powerful models such as context-free grammars and beyond are needed. The parameter estimation techniques for HMMs have been extended to these more powerful models also [31–33].

We will first give a brief description of the HMMs based on ref. 30. In a finite state model the state sequence can be determined (i.e. it is visible) from the sequence of the letters that are emitted when a state transition takes place. The letter sequence is observable. Thus from the observed sequence the state sequence can be determined. In contrast, in an HMM, in each state transition, a particular letter will be emitted according to some probability density function. Thus the state sequence cannot be unambiguously determined from the letter sequence, i.e. the state sequence is hidden, and hence the name HMM.

In order to use the HMMs first the model has to be trained, i.e. the parameters of the model have to be estimated using a set of training data. The training consists of first aligning the training data to the model and then reestimating the parameters of the model. This method is called the forward-backward (or Baum-Welsh) method. A simple description of this method and how HMMs are used in speech recognition appear in ref. 30.

HMMs are equivalent to finite state (stochastic) grammars (regular grammars). Finite state grammars are not adequate to model certain aspects of language, in particular the recursive aspects, as described earlier.

Hence, it is useful to consider more powerful grammars such as context-free grammars, i.e. consider stochastic context-free grammars. The forward-backward algorithm for training HMMs can be extended to stochastic context-free grammars[31-33]. In this case, it is often referred to as the inside-outside algorithm. We assume that the context-free grammar is in the Chomsky normal form, i.e. the rules of the grammar are of the form

$$A \rightarrow BC$$

$$A \rightarrow a$$

where A, B, and C are nonterminals in the grammar and a is a terminal symbol. Let $w = a_1, a_2, \cdots, a_n$ be the string of words (observation sequence). Training this model consists of determining a set of grammar rules given a training set of sentences (strings of words), $w_1, w_2, \cdots, w_n$. Instead of computing the forward and backward probabilities as in the case of HMMs, we compute inside and outside probabilities. Very roughly the inside probability is a computation that proceeds from bottom to top in the derivation tree and the outside probability computation proceeds from top down in the derivation of a string. For a simple description of this algorithm and its use in the reestimation of the parameters (the probabilities associated with the rules), see refs. 31–33.

More recently, a similar inside-outside algorithm for reestimation has been designed and implemented for the tree adjoining grammars[34]. Unlike the reestimation algorithm for HMMs, whose complexity of computation is $O(n)$, where n is the length of the input string (observation sequence), the complexities of the reestimation algortithms for the context-free grammars and tree adjoining grammars are $O(n^3)$ and $O(n^6)$ respectively. This increased complexity has not made these models applicable in practice yet. However, research is currently under way to make the computations more efficient[31].

We will now give a few examples to show how structural and statistical information can be integrated. Context-free grammars (CFG) have been used extensively in modeling grammars. Each rule (production) in a CFG can be associated with a probability of its use. Thus, given a CFG with rules: (R1) S→NP VP (0.9), (R2) S→NP NP V (0.1), (R3) VP→V NP (0.7), (R4) VP→V (0.3), we have associated probabilities with each of the rules. The probabilities of all rules associated with a given non-terminal add up to 1. The probability of a sentence (more precisely the derivation of the sentence in the grammar) is simply the product of the probabilities of each rule in the derivation because the grammar is CFG and the application of a rule depends only on the non-terminal on the left hand side of a rule and not on the context in which this non-terminal appears in a derivation. Probabilistic parsing methods and methods for estimating the probabilities of the rules from a training corpus are given in refs. 31–34. By making the probability associated with each rule somewhat context-dependent, for example, making it dependent on the preceding rule in the derivation, considerable improvement in the estimation of the probabilities and performance of the parser (in terms of getting correct parsers) can be achieved[35].

As we have seen earlier, the really 'meaningful' statistical dependencies are between words (lexical items) mediated most likely by grammatical relations. For example, there will be 'meaningful' statistical dependencies between the verb *eats*, and the lexical items that can appear as subject and object of *eats*. CFGs and their generalizations are not directly based on lexical items, that is, they are not, and in general, cannot be lexicalized[13]. Lexicalized grammars, as described earlier, are more appropriate for integrating structural and statistical information in a uniform manner.

Two dependent words in a sentence can be at an arbitrary distance apart, as we have seen earlier. Hence, this dependency cannot be captured by one-word, two-word, three-word and n-word frequencies, for some fixed n (that is, uni-gram, bi-gram, tri-gram and n-gram statistics). However, in many situations these statistics work surprisingly well in determining some aspects of language structure. Tri-gram frequencies (of parts of speech — that is, syntactic categories — and not words directly) have been used very successfully for discovering an optimum assignment of parts of speech to words[36,37]. Almost all words are lexically ambiguous, that is they belong to more than one category. For example, *table* is either a noun (N) or a verb (V); *pale* is either an adjective (ADJ) or an adverb (ADV); *see* can be a verb (V), an interjection (UM), or a noun (with capital S); *round* can be an adjective (ADJ), noun (N), verb (V), or an adverb (ADV), and so forth. The program in ref. 36 uses a linear time dynamic programming algorithm to find an assignment of parts of speech optimizing the product of: (i) probability of observing a part of speech i, given the word j, and (ii) probability of observing part of speech i, given two previous parts of speech. Probability estimates are obtained by training on a tagged corpus (such as the well-known Tagged Brown Corpus[38]). Error rates of only 3% to 4% have been reported[36], which compare very well with the error-rate of human annotators. Similar techniques have been used to locate simple noun phrases with high accuracy[36].

Statistical techniques in conjunction with large corpora (raw texts or annotated in various ways) have

also been used to automatically acquire other linguistic information such as morphological information (that is, parts of words such as prefixes and suffixes and inflected forms), subcategorization information (see the earlier section on grammars and parsers for subcategorization information), semantic classes (such as classification of nouns, based on what predicates they go with; compound nouns such as *jet engines, stock market prices*; classification of verbs, for example, *to know* describes a state of the world, while *to look* describes events and so on), and, of course, grammatical structure itself as we have already mentioned[35,39–43]. Such results have opened up a new direction of research in NLP, which is often described as corpus-based NLP.

It should be clear from the previous discussion that, for the development of corpus-based NLP, very large quantities of data are required (the Brown Corpus from the sixties is about 1 million words). Researchers estimate that about 100 million words will be required for some tasks. The technologies that will benefit from corpus-based NLP include speech recognition and synthesis, machine translation, full-text information retrieval, and message understanding, among others. The need for establishing very large text and speech databases, annotated in various ways is now well understood. It is recognized that no single organization can afford to create enough linguistic data even for its own research and development, let alone for the needs of the research community at large. This need, together with the size of the database and the need for sharing it, has been the key motivation for the plans for setting up a Linguistic Data Consortium (LDC) by DARPA[44]. Initial plans of the LDC call for the collection of raw text (naturally occurring text from a wide range of sources, 5 to 10 billion words); annotated text (syntactic and semantic labeling of some parts of raw text, upwards of 20 million words); raw speech (spontaneous speech from a variety of interactive tasks, 400 hours, 2000 speakers); read speech (1000 hours, 10,000 speakers); annotated spech (phonetic and prosodic labeling, 20 hours); a lexicon (a computational dictionary of 200,000 entries plus a term bank containing, for example, geographical, individual, and organizational names, 200 to 300 thousand entries); and a broad coverage computational grammar. The LDC will also develop a variety of sharable tools. Some examples in the speech area are: programs for segmentation of speech, alignment of speech and text, prediction of pronunciation options from orthographic transcription. Some examples from text are: a program for breaking text into sentences, a statistical parts-of-speech tagger, an efficient program for computing *n*-gram statistics and a variety of other statistics over very large corpora[44].

Integration of language and graphic/animation modalities

In this section*, I will describe a system that integrates language and graphics/animation modalities in the context of following task instructions in a changing (but hospitable) environment. This system (called AnimNL: Animation from Natural Language) has been developed by Badler and Webber and their associates at the University of Pennsylvania.

This work builds on the premise that agents rarely know everything they need to: they may not know exactly *how* to do things or what will happen when they attempt to do them. Instructions can help with both. The question is how agents can interact systematically with environments about which they have only partial relevant knowledge. The solution adopted in *AnimNL* is to enable agents, faced with incomplete knowledge, to

- develop expectations through instruction understanding and plan inference, and use those expectations, for example, in deciding how to act;
- exploit generalized abilities in order to deal with novel geometric situations.

The AnimNL system has as its goal the automatic creation of *animated task simulations* from *natural-language instructions*. The agents participating in these task simulations are animated human figures. AnimNL is intended to support advanced human factors analysis in what has come to be called *virtual prototyping*, enabling users of computer-aided design tools to simulate people's interactions with the artifacts they are designing and thereby to notice design flaws that might otherwise only become apparent after the artifacts enter the workplace.

In this system we assume that agents and artifacts are in fairly 'hospitable' environments—that is, ones that don't change drastically from moment to moment outside the agent's control. The point is to see if and how a task can be carried out with everything going fine. However, even in hospitable environments, classical planning techniques are insufficient for agents with limited knowledge. An agent cannot know what is in a closed, opaque box until he opens it, or what is needed to open a door until he knows what kind of door it is and if and how it is locked, or how many chopping motions he will need to chop an onion until

*This section is entirely based on several sections (edited for the current purpose) of a draft version of a paper by Bonnie Webber *et al.* (1992), *Doing What You're Told: Following Task Instruction in Changing, but Hospitable Environments*, Technical Report, Department of Computer and Information Science, University of Pennsylvania, 1992.

he sees that it is chopped enough for current purposes. Agents cannot, in general, know all that will be needed to complete a job until the job is complete.

The AnimNL system builds on an animation system, *Jack*™, (developed at the Computer Graphics Research Lab at the University of Pennsylvania). *Jack* provides articulated, animated human figures capable of realistic motion through model-based behaviors. In addition, *Jack* agents can be anthropometrically sized and given different 'strengths', so as to vary their physical capabilities. Different spatial environments can be constructed and modified at will, so as to vary the situations in which tasks are carried out. Such flexibility enables designers to explore a wide range of usage situations.

The AnimNL system addresses the problem of how to map from *high-level task specifications* (specifying a structure of related goals to be achieved and constraints—positive and negative—on how to achieve them) to *plausible physical behaviors* performed veridically. Generating and animating the behavior of highly articulated agents carrying out tasks in a physically veridical manner, in an environment about which they have only partial knowledge, raises a number of problems that have not received serious consideration in planning research or in robotics, although the solutions in Animation draw upon important recent work in Natural Language semantics[45], planning and plan inference[46-48], philosophical studies of intention[49], reasoning about knowledge and action[50,51] and subsumption architectures for autonomous agents.

Intentions as a factor in behavior

While AnimNL demonstrates agents' use of expectations and generalized abilities to deal with an environment they can know only partially, these are not the only factors influencing their behavior. In order to get agents to behave veridically, it is necessary to push notions of intention down from the highest cognitive level, to levels of basic movement.

Consider the following situation: An agent is in a house, in a room known to be adjacent to the kitchen. The door between the two rooms is closed. The agent is given the instruction 'Go into the kitchen to get me the coffee urn'. This instruction leads the agent to adopt intentions of going to the door to the kitchen and opening it. What is important is that the system cannot make low-level decisions about how the agent should move in doing either, without taking into account their intentions: they have implications concerning how close to get to the door, where to stand with respect to the door, and how wide the door should be opened.

Intention has been identified as a modulating factor in rational behavior by various researchers (e.g., refs. 48,49,52). However, the role of intentions in interpreting instructions has not yet been fully explored. Chapman[52] does stress that an instruction such as 'use the knife' can only be carried out after it has been interpreted *in terms of the situation at hand*. However, for Chapman, intentions are not purposes that an agent adopts, but rather features of the current situation that make a particular course of action sensible: in the context of Chapman's video-game, a knife could be used either to kill a monster or to jimmy a door. The situation itself—e.g. a monster is threatening the agent—will make clear what actions are sensible and determine the 'right' sense of 'use the knife'. Such a grounding of action in performance, and therefore acting in ways that make sense in a certain situation, is also embodied in AnimNL. However, the notion of intention has been generalized to cover the purposes that agents actively invoke while acting or planning to act.

Overview of AnimNL

The AnimNL system architecture reflects what has been found to be *necessary* to enable an agent to understand and act in accordance with purposeful instructions and to enable an animation system to simulate and animate that behavior. The architecture consists of two relatively independent kinds of processes. The first kind produces commitments to purposeful activity, which we call *annotated task actions*—for example,

- goto(door1, open (door1))–'go to door1 for the purpose of opening it'
- grasp(urn1, carry(urn1))–'grasp urn1 for the purpose of carrying it'.

The second kind of process determines how the agent should *move* in order to fulfil these commitments. What we will describe here is how instructions lead to initial commitments to act and how actions once embarked upon allow further commitments to be made and acted on.

Instructions are given to AnimNL in *steps* consisting of one or more utterances. Such multiclause instruction steps are common in maintenance and assembly instructions—for example,

> 'With door opened, adjust switch until roller contacts cam and continuity is indicated at pins A and B. Verify positive switch contact by tightening bottom nut one additional turn'. (Air Force manual T.O. 1F–16C–2– 94JG–50–2, p. 5–24)

While there are no firm guidelines as to what a single instruction step should encompass, often steps are organized around small coherent sub-tasks (such as

adjusting a switch). Such a step may specify several actions that need to be performed together (possibly in some partially specified order) to accomplish a single subtask, or several aspects of a single complex action (e.g. its purpose, manner, things to watch out for, appropriate termination conditions, etc.). The agent must develop some degree of understanding of the whole step before starting to act.

A 'step', for AnimNL, not only defines a subtask, but also specifies behavior that the agent must attend to continuously: while carrying out a step, attention must be maintained on the task at hand. The agent cannot stop until he finishes or hits a snag in the current step. In other words, a step defines the sequence of the instructions that must be processed as a whole, *before* the agent begins to act on them. The example instruction step that we will focus on and illustrate in this article is 'Go into the kitchen to get me the coffee urn'. While it contains only two clauses, it does illustrate a surprisingly large number of interesting points.

Steps are processed by a natural language parser to produce an action representation based on Jackendoff's *Conceptual Structures*[45]. For example, from the input instruction 'Get me the coffee urn', the parser produces:

$$[\text{CAUSE}(i, [\text{GO}_{\text{Sp}}([\text{URN-OF-COFFEE}]_{,}, k)])]_{\beta}$$

$$\begin{bmatrix} \text{FROM}([\text{AT}(j)]) \\ \text{TO}\ (me) \end{bmatrix}$$

This can be glossed as *the agent causing the coffee urn to go from where it is now to where the speaker is.*

One reason for using Jackendoff's Conceptual Structures is that they can reveal where information may be missing from an utterance and have to be provided by inference. Here the representation makes explicit where the coffee urn must be moved from, namely 'AT([j])'—where the urn is now. (If the sentence had been 'get me the coffee urn from the kitchen', 'the kitchen' would fill this argument.)

The indexed conceptual structures corresponding to a single instruction step are incrementally developed into a *plan graph* that represents the structure of the animated agent's intentions, via processes of *reference resolution, plan inference, reference grounding* and *plan expansion.* The leaves of the plan graph are *annotated task actions,* such as those shown at the beginning of this section.

The incremental nature of plan graph development derives, in part, from limits on agent knowledge. In particular, we assume that an agent cannot have up-to-date *knowledge* of any part of its environment that is outside its direct perception, where its direct perception is limited by a *portal assumption:* Portals like doors connect opaquely bounded three-dimensional subspaces, which may be adjacent to or embedded within one another (e.g., boxes inside boxes). Agents can only see into, and hence only know the contents of, spaces that have a *portal* open into the space the agent occupies, which we call the *active space.* Thus agents have to open doors, lids, etc. to explore other spaces. Within the *active space,* we make the simplified assumption that agents can see everything up to closed portals.

When an annotated task action (i.e. a leaf in the plan graph) becomes sufficiently specified for the agent to be ready to commit to it and temporal dependencies between task actions permit such commitment, it will be gated, triggering object-specific reasoning and processes associated with simulating the agent's movements and other environmental activity. More specifically, an annotated task action will be gated when

- the action is 'executable'.
- all actions temporally prior to it have been committed to (Previous action need not have been completed: an agent can be, and usually is, doing several things at one time.)
- its purpose has been determined.

The output of these low-level processes may either be an indication that the agent is unable to carry out the desired actions or a 'program' of *behaviors* to be executed (simulated) in parallel. Actions change the world, as well as the agent's knowledge. Such changes trigger further elaboration of the agent's intentional structure (i.e. the plan graph) and further commitments to action. In the next section, we will focus on aspects of instructions that lead to *expectations* on the part of the agent.

Expectations from instructions

Usually one thinks of instructions in terms of what they explicitly tell one to do or to avoid doing. Such structures may be conveyed explicitly or implicitly. In AnimNL, the focus is on a different role that instructions can play—that of providing agents with specific, useful *expectations* about actions and their consequences.

We noted in our discussion of instruction *steps* in the previous section, that steps may specify several actions that need to be performed together, often under partial ordering, to accomplish a single sub-task. Agents must develop some level of understanding of an entire step before beginning to act. This is the most obvious sense in which instructions set up agent expectations: by assuming that an instruction step reliably describes a

way of accomplishing a sub-task, an agent will expect that the conditions needed for accomplishing subsequent parts of the specified sub-task will hold when they need to. The agent will not expect that he needs to do anything else in order to make these conditions hold.

Besides these general expectations, we can identify three more specific types of expectations that instructions may engender, that agents can use in carrying out their tasks, as follows:

- expectations about how long a process will take to come to some desired state;
- expectations about the intended consequences of an action;
- expectations about where objects are to be found.

These are discussed in turn in the next three subsections.

Expectations about processes

In some earlier work designed for creating animations from recipes, Karlin[53] analysed a range of temporal and frequency adverbs found in instructions. One particular construction she analysed is the following:

Do α for < duration > or until < event >

e.g. 'Steam 2 minutes or until mussels open'.

Karlin notes that this is not a case of logical disjunction, where the agent can choose which disjunct to follow: rather, the explicit duration suggests the usual amount of time that it will take the mussel-steaming process to effect the desired change in the mussels' state. The desired state is that all the mussels that were closed when they were put into the pot (already open ones having been discarded as dead, prior to this point) are now open. If they are not open after two minutes, the agent should wait a bit longer. Those that have not opened after another short wait should then be discarded, since they contain nothing but mud.

The usefulness of this expectation of how long a process will take to come to some desired state comes from the *cost* of sensing. In the case of steaming, cooking is usually done in a closed, opaque cooking pot. Every time the lid is removed to check the state of the contents, the steam used to cook the contents escapes, setting the process back. The result of sensing too often, is that the mussels become tough through over-cooking. The expectation can therefore be used by the agent to gauge how long he can safely wait before beginning to make costly sensing tests.

Expectations about consequences

There are expectations about the intended properties of objects produced by actions that can have any of several results. Consider for example, the action of mixing flour, butter and water. Depending on the relative amounts of these three ingredients and the porosity of the flour, the result may be anything from a flaky mass to a viscous batter. Instructions may tell an agent what the intended result is, so that he or she can augment the amount of one ingredient or another, if the immediate result does not satisfy the intended description. For example,

a. Mix the flour, butter and water, and *knead* until smooth and shiny.

b. Mix the flour, butter and water, and *spread* over the blueberries.

c. Mix the flour, butter and water, and *stir* until all lumps are gone.

Here the verbs 'knead', 'spread' and 'stir' convey the expected viscosity of the resulting mixture.

Expectations about locations

Because multi-clause instruction steps may evoke more than one situational context, part of an agent's cognitive task in understanding an instruction step is to determine that situation in which he is meant to find a referent for each of its referring expressions. This is the process of *grounding* referring expressions. Because actions can effect changes in the world or in what is visible to an agent, certain referring expressions in an instruction step might refer to the current world and what the agent is aware of within it, while other expressions might refer to objects that may only come into existence, or whose existence the agent may only become aware of, in the future.

To see this, compare the two instructions

a. Go into the kitchen to get me the coffee urn.

b. Go into the kitchen and wash out the coffee urn.

In the first instruction, 'the coffee urn' will generally be taken to denote an urn currently in the kitchen, one that the agent, when given the instruction, may not even be aware of. When given this instruction agents appear to develop an expectation that *after* they perform the action, they will be in a context in which it makes sense to try to ground the expression and determine its referent.

The second instruction ('... and wash out the coffee urn') is different: if the agent sees a coffee urn in the current context, prior to acting, he will happily ground the referring expression 'the coffee urn' against that object. If he does not see a coffee urn prior to acting, he

develops the same expectation as in the first example, that when he gets into the kitchen, he will be able to ground the expression then. The fact that agents will look around when they get to the kitchen if a coffee urn is not immediately visible, opening cabinets until they find one, shows the strength of this expectation and the behavior it leads to.

This decision as to the context to use in grounding a referring expression is based on distinguishing the information (and assumptions) used to *resolve* a referring expression from that used to *ground* it. Reference *resolution* precedes reference *grounding* and involves using information from the interpretation of the current utterance (i.e. the explicit description), information from the previous discourse, and hypotheses about the intended relationship between actions.

The system has two modalities—the Action Library (or the Action Knowledge Base) and the plan graph. Since our focus is on the natural language aspect of the system (and also due to the limited space available in this paper), we will not describe these modalities. The modalities are used to compute the expectations. Thus while processing the instruction, 'Go into the kitchen to get me the coffee urn', using the Action Library and the Plan Graph, the system computes the expectation,

'The urn of coffee is in the kitchen'.

Once the instruction is understood in this way the appropriate actions are incorporated into the plan graph.

The difference between this instruction and another instruction 'Go into the kitchen to wash the coffee urn', where a currently visible urn would be taken as the most likely referent of 'the coffee urn'. Then in this case, the assumption *the urn of coffee is in the kitchen* will not be computed. Another assumption such as *the washing-materials are in the kitchen* will be computed.

In summary, we have described some ways in which instructions provide agents with useful expectations. With experience, human agents internalize general expectations and not rely on instructions for them. For AnimNL, this means incorporating expectations into its action schemata and using them in elaborating the plan graph. We omit here the description of other major components of the systems such as the planner—geometric and functional planner and the simulator. One goal here is to give some idea about the rich domain of natural language instructions for animated agents. This rich domain for natural language understanding is much more tractable than text understanding or story understanding. Also, integrating natural language and animation in this manner has several important applications, for example, the development of interactive maintenance manuals.

Summary

I have briefly described three areas in NLP which are currently very active and represent quite a diverse range of issues and methodologies. As regards to future prospects for NLP, clearly the trend of integrating speech and language will continue and we can expect much robust spoken language systems, which also incorporate some aspects of discourse structure. More advanced and useful commercial systems for multilingual interfaces, machine translation, and message understanding systems will appear in the near future. Speech-to-speech translation systems will also appear in limited domains. On the theoretical side, future mathematical and computational work will provide us more unifying accounts of syntax, semantics, and pragmatics. It will also contribute to psycholinguistic research which studies human language processing.

1. Grosz, B. J., Sparck-Jones, K. and Webber, B. L., *Readings in Natural Language Processing*, Morgan Kauffman, Los Gatos, CA, 1986, Chapters III and IV.
2. Chomsky, N., On certain formal properties of grammars, *Inf. Control*, 1959, **5**, 137–167.
3. Shieber, S., *An Introduction to Unification-Based Grammars, Lecture Notes, no. 4*, Center for Studies in Language and Information (CSLI), University of Chicago Press, Stanford University, 1986.
4. Gazdar, G., Klein, E., Pullum, G. K. and Sag, I. A., *Generalized Phrase Structure Grammars*, Harvard University Press, Cambridge, MA, 1985.
5. Pollard, C. and Sag, I. A., *Information-Based Syntax and Semantics*, Center for the Study of Language and Information (CSLI), Stanford University Press, Stanford University, 1986, vol. 1.
6. Kaplan, R. M. and Bresnan, J., Lexical functional grammar, in *The Mental Representation of Grammatical Relations*, (ed. Bresnan, J.), MIT Press, Cambridge, MA, 1983, pp. 173–281.
7. Shieber, S., Evidence against the context-freeness of natural language, *Linguist. Philos.*, 1985, **8**, 333–343.
8. Bresnan, J. W., Kaplan, R. M., Peters, P. S. and Zaenen, A., Cross-serial dependencies in Dutch, *Linguistic Inquiry*, 1982, **13**, 613–635.
9. Higginbotham, J., English is not a context-free language, *Linguistic Inquiry*, 1984, **15**(225), 225–234.
10. Culy, C., The complexity of the vocabulary of Bambra, *Linguist. and Philos.*, 1985, **8**, 345–351.
11. Joshi, A. K., How much context-sensitivity is necessary for characterizing structural descriptions—Tree adjoining grammars, in *Natural Language Processing: Theoretical, Computational and Psychological Perspectives* (eds. Zwicky, D., Dowty, D. and Karttunen, L.), Cambridge University Press, Cambridge, England, 1985.
12. Joshi, A. K., An introduction to tree adjoining grammars, *Mathematics of Language* (ed. Manaster-Ramer, A.), John Benjamin, Amsterdam, 1987, pp. 87–114.
13. Joshi, A. K. and Schabes, Y., Flexible phrase structure and coordination, in *Proceedings of DARPA Workshop on Spoken Language Systems*, Morgan Kauffman, Palo Alto, CA, 1991, pp. 195–199.
14. Steedman, M., Combinators and grammars, in *Categorial Grammars and Natural Language Structures*, Foris, Dordrecht, 1986.

15 Steedman, M., Combinatory grammars and parasitic gaps, *Nat. Language Linguist and Theor.*, 1987, 5, 403–439.
16 Kroch, A. S., Subjacency in a tree adjoining grammar, in *Mathematics of Language* (ed. Manaster-Ramer, A.), John Benjamin, Amsterdam, 1987, pages 143–172.
17 Kroch, A. S., Asymmetries in long-distance extraction in a TAG grammar, in *New Conceptions of Phrase Structure* (eds Baltin, M. and Kroch, A. S.), University of Chicago Press, Chicago, IL, 1989, pp. 66–98.
18 Kroch, A. S. and Santorini, B., The derived constituent structure of the West Germanic verb-raising constructions, in *Proceedings of the Princeton Workshop on Grammar* (ed. Freiden, R.), MIT Press, Cambridge, MA, 1991.
19 Joshi, A. K. Vijay-Shanker, K. and Weir, D., The convergence of mildly context-sensitive grammar formalisms, in *Processing of Linguistic Structure*. (eds. Shieber, S. and Wasow, T.), MIT Press, Cambridge, MA, 1991.
20. Gazder, G., *Applicability of indexed grammars to natural* languages, Technical Report CSLI 85–34, Center for the Study of Language and Information (CSLI), Stanford University, 1985.
21. Pollard, C., *Lecture Notes on Head-Driven Phrase Structure Grammars*, Center for the Study of Language and Information (CSLI), University of Chicago Press, Stanford University, 1985.
22. See *Proceedings of the DARPA Workshops on Spoken Language Systems*, Morgan Kauffman, Palo Alto CA, 1989, 1990, 1991.
23. Marcus, M., *A Theory of Syntactic Recognition for Natural Language*. MIT Press, Cambridge MA, 1980.
24. Hindle, D., User manual for Fidditch, Technical Memorandum #7590–142, Naval Research Laboratory, 1983.
25. Brill, E., Magerman, D., Marcus, M. and Santorini, B., Deducing linguistic structure from the statistics of large corpora, in *Proceedings of DARPA Workshop on Spoken Language Systems*, Morgan Kauffman, Palo Alto, CA, 1990, pp. 275–282.
26. Shannon, C. E., A mathematical theory of communication, *Bell Systems Tech. J.*, 1948, **27**(37a).
27. Chomsky, N., *Syntactic Structures*, Mouton and Co., S. Gravenhage, 1957.
28. Harris, Z. S., Co-occurrence and transformation in linguistic structure, *Language*, 1957, **3**, 283–340.
29. Baum, L. F. and Petrie, T. Statistical inference for probabilistic functions of finite state markov chains, *Annals of Mathematical Statistics*, 1966, **37**, 1554–1565.
30. Paul, D. B., Speech recognition using the hidden markov model, *The Lincoln Lab. J.*, 3(1), 41–62, Spring 1990.
31. Pereira, F. and Schabes, Y., Inside-outside reestimation from partially bracketed corpora, in *Proceedings of the DARPA Speech and Natural Language Systems Workshop*, Arden House, NY, Morgan Kauffman, Palo Alto, CA, 1992.
32. Lari, K. and Young, S. J., Applications of stochastic context-free grammars using the inside-outside algorithm, *Computer Speech Language*, 1990, **5**, 237–257.
33. Jelinek, F., Lafferty, J. D. and Mercer, R. L., Basic methods of probabilistic grammars, Technical report, IBM, Yorktown Heights, NY, 1990.
34. Schabes, Y., A inside–outside algorithm for estimating the parameters of a hidden stochastic context-free grammar based on Earley's algorithm, in *Second Workshop on Mathematics of Language (MOL)*, Yorktown Heights, NY, May 1991.
35. Magerman, D. and Marcus, M., Pearl: A probabilistic chart parser, in *Proceedings of the Fifth Conference of the European Association for Computational Linguistics*, Association for Computational Linguistics, Morristown NJ, 1991, pp. 15–20.
36. Church, K. W., A stochastic parts program and noun phrase parser for unrestricted text, in *Proceedings of the 2nd Conference on Applied Natural Language Processing*, Association for Computational Linguistics, Morristown, NJ, 1988, pp. 136–143.
37. DeRose, S., Grammatical category disambiguation by statistical optimization, *Comput. Linguist*, 1988, **14**(1).
38. Francis, W. and Kucera, H., *Frequency Analysis of English Usage: Lexican and Grammar*, Houghton Mifflin Company, Boston, MA, 1982.
39. Brill, E., Personal communication.
40. Brent, M. R. and Berwick, R., Automatic acquisition of subcategorization frames from tagged text, in *Proceedings of DARPA Workshop on Spoken Language Systems*, Morgan Kauffman, Palo Alto, CA, 1991, pp. 342–345.
41. Brent, M. R., Automatic semantic classification of verbs from their syntactic contexts: An implemented classifier for stativity, in *Proceedings of the Fifth Conference of the European Chapter of the Association for Computational Linguistics*, Association for Computional Linguistics, Morristown NJ, 1991, pp. 222–226.
42. Hindle, D., Noun classification from predicate-argument structures, in *Proceedings of the Association for Computational Linguistics Conference*, Association for Computational Linguistics, Morristown, NJ, 1990, pp. 268–275.
43. Smadja, F. and McKeown, K., Automatically extracting and representing collocations for language generation, in *Proceedings of the Association for Computational Linguistics Conference*, Association for Computational Linguistics, Morristown, NJ, 1990, pp. 252–259.
44. Liberman, M., Guidelines for the linguistic data consortium, Draft Proposal for DARPA, May 1991.
45. Jackendoff, R., *Semantic Structures*, MIT Press, Cambridge, MA, 1990.
46. Agre, P., Chapman, D. and Pengi, D., An implementation of a theory of activity, In *Proceedings of the AAAI-87 Conference*, June 1987, pp. 268–272.
47. McDermott, D., Planning and acting, *Cognit. Sci.*, 1978, **2**, 71–109.
48 Pollack, M., Inferring domain plans in question-answering, Technical report, Ph.D. thesis, Department of Computer and Information Science, Technical Report MS–CIS–86–40, University of Pennsylvania, 1986.
49. Bratman, M., *Intentions, Plans and Practical Reason*, Harvard University Press, Cambridge, MA, 1987.
50. Moore, R. C., Formal theories of the commonsense world, in *A Formal theory of Knowledge and Action* (eds. Moore, R. C. and Hobbs, J.), Ablex Publishing, Norwood, NJ, 1984.
51. Morgenstern, L., Knowledge preconditions for actions and plans, in *Proceedings of IJCAI*, Milano, Italy, June 1987, pp. 867–874.
52. Chapman, D., *Vision Instruction and Action*, MIT Press, Cambridge, MA, 1991.
53. Karlin, R., Defining the semantics, of verbal modifiers in the domain of cooking tasks, in *Proceedings of the 26th Annual Meeting, Association for Computational Linguistics*, Association for Computational Linguistics, Morristown, NJ, 1988, pp. 61–67.