Computers and the Humanities **22** (1988) 99—110.
© 1988 *by Kluwer Academic Publishers.*

The Use of Natural Language Processing in Computer-Assisted Language Instruction

Alan Bailin and Philip Thomson

Department of Effective Writing, University of Western Ontario, London, Ont. Canada, N6A 3K7

Abstract: This article describes the natural language processing techniques used in two computer-assisted language instruction programs: VERBCON and PARSER. VERBCON is a template-type program which teaches students how to use English verb forms in written texts. In the exercises verbs have been put into the infinitive, and students are required to supply appropriate verb forms. PARSER is intended to help students learn English sentence structure. Using a lexicon and production rules, it generates sentences and asks students to identify their grammatical parts. The article contends that only by incorporating natural language processing techniques can these programs offer a substantial number of exercises and at the same time provide students with informative feedback.

Key Words: natural language processing, sentence generation, grammar exercises, sentence structure, language instruction, computer-assisted language instruction, computer-assisted language learning, computer-assisted instruction, templates.

This paper will show how natural language processing techniques are used in two drill-and-practice CAI programs for English grammar instruction. The programs were developed for use by the Department of Effective Writing at The University of Western Ontario. The department offers workshops and tutorials in grammar and rhetoric. More than three hundred students a year use its services. Many of these students have difficulty writing standard English.

To acquire the grammatical skills necessary for writing standard English, students generally need a great deal of practice. Such practice should involve immediate feedback to reinforce appropriate responses and discourage inappropriate ones. Moreover, the feedback should involve more than just an indication of whether a response is right or wrong. It should help students to understand why an answer is or is not acceptable so that they can generalize from the learning experience.

Although Effective Writing does give individualized attention in its workshops and tutorials, its staff is small and this limits both the number of students who can be helped and the time which can be spent on each one. Computer-assisted instruction seemed a way to provide individualized instruction. We chose to develop CAI packages for two areas where the instruction was weakest: the appropriate use of verb forms and the analysis of English sentence structure.

Many of the students who use the department's services (particularly those whose first language is not English) have difficulty employing appropriate verb forms. The difficulty extends beyond tense; the students in question also have difficulty with mood, voice, aspect and person/number. Although drill-and-practice is often useful in teaching students the pertinent skills, such exercises require a substantial commitment of instructional time. VERBCON, the first of the CAI programs we developed, has allowed the department to give students considerably more instruction in this area than was possible previously. It presents students with sentences and texts in which many verbs are in the infinitival form and asks them for forms which are appropriate in the context.

CAI performs a similar function in relation to our teaching of English sentence structure. Many

Alan Bailin is director of the Effective Writing Program at the University of Western Ontario, London, Ontario, Canada. Philip Thomson is a programmer in the Faculty of Medecine, University of Western Ontario.

of our students have little or no familiarity with the grammatical rules for written English. In order to teach students these rules, we first try to acquaint them with English sentence structure and a set of grammatical terms (e.g., "noun", "verb", "prepositional phrase") with which to describe it. Here again drill-and-practice can be an important tool: the more students can practice parsing sentences, the more comfortable they become with the basic concepts and terms. However, like verb-form exercises, parsing assignments can consume considerable instructional time. CAI seemed a good means of increasing the department's capacity in this area as well. PARSER, the second of the programs we developed, provides that increased capacity: it gives students practice identifying the grammatical parts of sentences it generates.

We could have developed the type of software in which both questions and answers are fixed. An example of this approach is found in CLEF (1985—87; see Holmes 1983 and Burghardt 1984: 165). CLEF consists of a set of drill-and-practice lessons to teach French grammar. Although these exercises provide informative linguistic feedback for students, the feedback is in general tailored to the specific exercise items. Consequently, detailed information is required for each possible response to each question. For this reason, such custom programming is very time-consuming.

A "traditional" template system, on the other hand, would have allowed us to create exercises in much less time. In a template system, the form of the exercises is fixed, but any number of exercises in the predetermined form can be created. James Pusack's DASHER (1982) is an example of the "traditional" form of the template system (see Pusack 1983 and Cully 1984). It can be used to create various types of drill-and-practice exercises for a wide range of languages. However, the feedback it gives is purely mechanical. It can identify such errors as misspellings and incorrect word order, but can give no information concerning the linguistic characteristics of the language itself: its feedback is essentially of a word-processing sort.

What we needed was software sophisticated enough to give students pertinent grammatical information, yet general enough to produce substantial numbers of exercises. The key to creating such software, we found, was to take advantage of the regularities in the language itself by using natural language processing techniques. VERBCON is a template-type program which can produce informative linguistic (as opposed to mechanical) feedback because it takes advantage of the fact that English verb forms involve fixed patterns. PARSER can not only give informative linguistic feedback but can also generate its own exercise items because it capitalizes on the fact that English sentences have recurrent structures.

Below we discuss each of the packages in turn. In each case we first describe the courseware itself and then show how natural language processing is used within it.

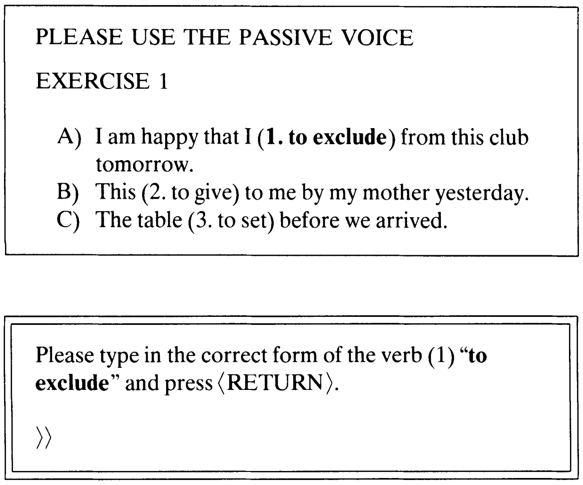
VERBCON

A Description

VERBCON is written in Microsoft Pascal 3.2 and runs on an IBM PC with at least 256k memory. Although a monochrome monitor can be used, the courseware does utilize colour and therefore a colour monitor is preferable. As noted above, the program's purpose is to help students master appropriate English verb forms in written contexts. To this end VERBCON is comprised of two parts: a grammar review and a set of exercises. The student has the choice of reviewing the grammar or immediately beginning the exercises.

The grammar review is concerned with the characteristics of English verb forms. More specifically, it gives explanations and examples of the tenses, aspects, moods, and voices in which English verbs can be used. In order to make it easy for students to view the material, key terms and verb forms are highlighted, and the review is divided into discrete screens, each of which discusses a distinct property of verb forms (e.g., a specific tense). Students can page through the material at leisure, spending as much or as little time as they wish on any particular part.

In the exercises verbs have been put into the infinitive, and students are asked to supply appropriate verb forms. Some of the exercises focus on a specific tense or voice (see Figure 1), while others present passages from essays in order to give students practice in using English verbs in


PLEASE USE THE PASSIVE VOICE

EXERCISE 1

A) I am happy that I (**1. to exclude**) from this club tomorrow.
B) This (2. to give) to me by my mother yesterday.
C) The table (3. to set) before we arrived.

Please type in the correct form of the verb (1) "**to exclude**" and press ⟨RETURN⟩.

⟩⟩

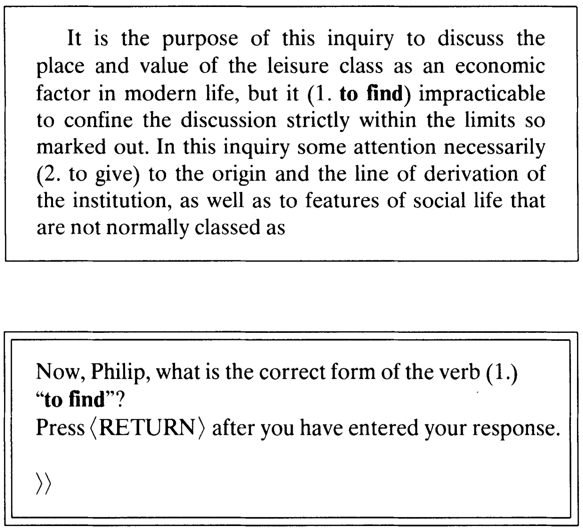
Figure 1. An example of an exercise focusing on a specific voice.

extended written contexts (see Figure 2). The number of exercises which can be included in VERBCON is not limited: the software is not tailored to a specific set of exercises.

The presentation of exercises is straightforward. Highlighting is used to indicate the current item (see Figure 2). To allow an exercise to extend beyond a single screen, VERBCON has a scrolling mechanism which permits students to view any part of the exercise at will. Nevertheless, each exercise operates in a sequential manner, beginning with the first item and proceeding systematically to the last.

Students are given three opportunities to give a correct response to each exercise item. After the third attempt, a correct answer is provided. Because more than one verb form can often function as an appropriate response, VERBCON allows up to three correct responses. If there is more than one correct response, however, one of them must be chosen as the "preferred response", that is, the answer which VERBCON will supply to a student after a third unsuccessful attempt.

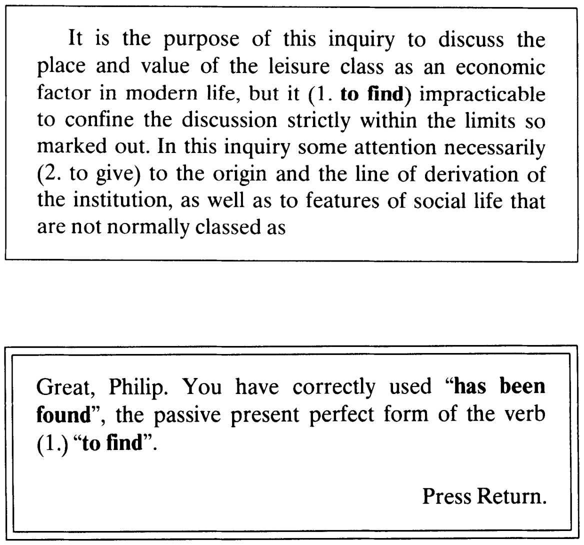
VERBCON has a number of answer-processing features. To familiarize students with the properties of English verb forms, VERBCON includes a procedure to tell students the kind of verb form they have used — whether or not their answer is correct. If a student types in "has been found" as the answer for the first item of the exercise in Figure 2, VERBCON not only reports that the form "has been found" is a correct answer; it also reports that this answer is a present perfect passive form of the verb "to find". Both the verb form and the information concerning it are highlighted (see Figure 3). So that students are not overloaded with details, the program supresses


It is the purpose of this inquiry to discuss the place and value of the leisure class as an economic factor in modern life, but it (1. **to find**) impracticable to confine the discussion strictly within the limits so marked out. In this inquiry some attention necessarily (2. to give) to the origin and the line of derivation of the institution, as well as to features of social life that are not normally classed as

Now, Philip, what is the correct form of the verb (1.) "**to find**"?
Press ⟨RETURN⟩ after you have entered your response.

⟩⟩

Figure 2. An example of an exercise involving an essay passage.


It is the purpose of this inquiry to discuss the place and value of the leisure class as an economic factor in modern life, but it (1. **to find**) impracticable to confine the discussion strictly within the limits so marked out. In this inquiry some attention necessarily (2. to give) to the origin and the line of derivation of the institution, as well as to features of social life that are not normally classed as

Great, Philip. You have correctly used "**has been found**", the passive present perfect form of the verb (1.) "**to find**".

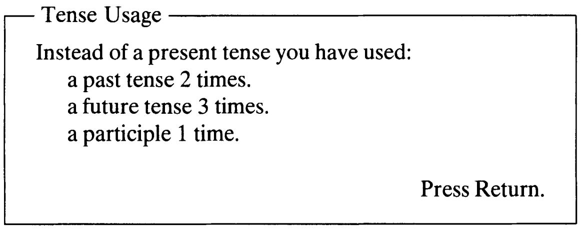
Press Return.

Figure 3. An example of the feedback for a correct response.

certain "default" information concerning forms. Although VERBCON indicates, for example, that a verb form is passive, it does not indicate a form is active. English speakers generally consider the active to be the normal voice of the verb. The use of the passive, however, is noteworthy, particularly in writing.

After each attempt, VERBCON gives students a hint about the nature of the correct answer, or the preferred correct answer if there is more than one possibility. Should the verb form given in the response have a different tense and voice than the correct answer, VERBCON will tell the student the correct voice; if the voice and aspect are incorrect, it will report the correct aspect. If, for example, a student types in "will find" as the answer for the first item of the exercise in Figure 2, VERBCON will suggest a past rather than a future tense; if he types in "is finding", it will recommend the perfect aspect. The kind of hint depends on the nature of the response.

At the end of each exercise, VERBCON supplies information concerning the nature of the student's errors. It will report the number of times each tense, mood, aspect, and voice has been misused, as well as the number of subject-verb agreement problems. A sample screen presentation of this diagnostic information is given in Figure 4. Students (and their teachers) can use the diagnostics to identify specific problem areas in verb usage.



```
Tense Usage
Instead of a present tense you have used:
    a past tense 2 times.
    a future tense 3 times.
    a participle 1 time.

                                  Press Return.
```

Figure 4. A typical diagnostic screen.

The Identification of English Verb Forms

Central to the operation of VERBCON is its ability to identify the forms of English verbs. This ability underlies all of VERBCON's answer-processing capabilities: the identification of the verb forms a student has used in both correct and incorrect answers, the hints concerning correct answers, and the error diagnostics. This capability also allows VERBCON to include exercises involving any English verb. If the verb identification routines were applicable only to specific verbs, the exercises which could be included in VERBCON would be severly limited. In this section we will describe the natural language processing techniques which give VERBCON its verb identification capability.

While conceivably one could simply list every form of every single verb one uses, this would be an exceptionally tedious and, perhaps more importantly, inefficient procedure. Ultimately simpler is a parsing routine which identifies verb forms on the basis of linguistic properties. VERBCON uses just such a procedure. In order to make its identifications, VERBCON needs ony the five basic forms of a verb: the infinitive, the third person singuar, the preterit, and the past and present participles. A parsing algorithm does the rest.

VERBCON uses a situation-action parsing algorithm (see Winograd 1983: 401ff). This kind of algorithm stipulates that certain "actions" are to be taken when certain "situations" are encountered. The "situations" VERBCON uses are, in fact, quite simple properties of verbs: number of auxiliaries, type of auxiliary, type of ending, etc. The "actions" it takes involve either looking for another property of the verb or attaching a grammatical label to the verb form (e.g., present progressive, simple past). For example, if VERBCON "sees" either the infinitive or third person singular form of "have" ("have" and "has" respectively), it "looks for" the past participle form of the verb. If it then finds the past participle form, it identifies the verb form as the present perfect. If, on the other hand, it "sees" the forms "was" or "were", it "looks for" the present participle and if it finds this form, it identifies the verb form as the past progressive. These identification procedures work for every single verb in English.

These are many ways of implementing the kind of identification techniques discussed above. From a conceptual point of view, perhaps the simplest is to identify salient characteristics and, on this basis, to attempt a guess at the identity of the complete structure. If the guess proves wrong one can simply backtrack and try to find some other characteristics on which to base another guess. However, this approach is somewhat inefficient: time is wasted when the program makes incorrect guesses and the computer must backtrack.

More efficient is a deterministic parsing routine (see Marcus 1980: 1—25; Winograd 1983: 409—410), that is, one which takes a set of actions that lead to an identification without backtracking (and without exploring more than one possibility at a time through parallel processing). The software we have developed for identifying verb forms does just this for all but single verb forms. The basic strategy is simple: attempt to find characteristics of the input which allow one progressively to narrow down the range of possibilities. The following is a selection from the algorithm for verb forms involving three words:

A. If the 1st word begins with the letter *h*, then call Condition A true.
B. If Condition A is true and the 3rd word ends in *ing*, then Condition B is true; otherwise Condition B is false.
C. If Conditions A and B are true and the 1st word is *have* or *has*, then the verb form is a present progressive.
D. If Conditions A and B are true and the first word is *had*, then the verb is a past progressive.
E. If Condition A is true and Condition B is false, then Condition E is true.
F. If Condition E is true and the first word is either *have* or *has*, then the verb is a present perfect passive.
G. If Condition E is true and the first word is *had*, then the verb is a past perfect passive; otherwise Condition G is false.
H. If Condition E is true and Condition G is false, then the verb is a perfect passive participle.

Each step in the parsing algorithm presented above is equivalent to an IF-THEN-ELSE step in the Pascal code. The Pascal implementation of the fragment of this parsing algorithm exemplifies the way the algorithm is realized in the program.

```
IF FIRSTWORD [1] = 'H' THEN
BEGIN
  IF RIGHT (THIRDWORD, 3) = 'ING'
  THEN BEGIN
                 IF FIRSTWORD = 'HAVE'
                 THEN MATCHFORM (14111);
                 ELSE IF FIRSTWORD = 'HAS'
                         THEN MATCHFORM (14311)
                         ELSE IF FIRSTWORD = 'HAD'
                                 ELSE MATCHFORM (14121);
          END
  ELSE IF FIRSTWORD = 'HAVE'
          THEN MATCHFORM (23111)
          ELSE IF FIRSTWORD = 'HAS'
                  THEN MATCHFORM (23311)
                  ELSE IF FIRSTWORD = 'HAD'
                          THEN MATCHFORM (23121)
                          ELSE IF FIRSTWORD = 'HAVING'
                                  THEN IF SECONDWORD = 'BEEN'
                                          THEN MATCHFORM (26131);
  END;
```

In this code the IF-THEN-ELSE statements identify the verb form. MATCHFORM then uses the identification to create a model with which the inputted form can be compared to insure against misspellings or other mistakes.

To see how the procedure actually works, let us assume that in answering the question presented in Figure 2, a student incorrectly types in "had been found". The first "IF" determines whether the first word starts with the letter "H". Since this is the case, the first auxiliary must be a form of "have" ("have", "has", "had", or "having") and the verb form must be in the perfect aspect. The next "IF" checks to see whether "had been found" is a present or past perfect progressive: it examines the third word (that is, the verb stem) for the "ING" ending. Since the third word, "found", does not end in "ING", the verb form cannot be either of these kinds of perfect progressives. By a process of elimination, the verb form must be a perfect passive. The tense (past, present, future) is determined by checking the first word again for "have" and "has" to establish whether the verb form is a present perfect passive. Since the verb from does not begin with either "have" or "has", the procedure looks for "had" to see if the verb form is a past perfect passive. The verb form does in fact begin with "had" so the procedure identifies "had been found" as a past perfect passive. MATCHFORM then creates the past perfect passive of the verb "to find" in order to insure that the student's input is indeed the past perfect passive form and not a construction which is ill-formed because of mistyping or some other kind of error. In creating the verb form, MATCH-FORM uses information that is employed in the parsing procedure itself — in this case, that the past perfect passive is formed by combining the auxiliaries "had" and "been" with the past participle of the verb.

The parsing procedure needs little information concerning individual verbs: only the infinitive, third person singular present, preterit, and present and past participles must be supplied. Without the built-in parsing routines, either an extremely large amount of information about individual verbs would be needed, or the program would have to do without the kinds of sophisticated feedback which it can supply. In the former case, lesson creation would be a laborious process. In the latter, VERBCON would be subject to the kind of criticism which Gerald Culley directs against "generic", template-style approaches; that is, that they achieve generality at the cost of informative feedback (Culley 1984: 183—185). However, by looking at the problem of giving feedback as a natural language processing problem involving the parsing of verb forms, we have been able to create software which is generic in form yet specific enough to give students instructive feed-back.

The deterministic approach allows VERBCON to identify verb forms in a fraction of a second. In addition to making the program more efficient, it has permitted the creation of a simple authoring system. In the current version, VERBCON not only uses the procedure outlined above to identify the student input; it also uses it to parse the correct answer with no noticeable increase in the time needed to give students feedback. The courseware developer thus only needs to supply the correct answer; VERBCON identifies its tense, mood, voice and aspect. This makes the development process simpler and prevents human error in the identification process.

PARSER

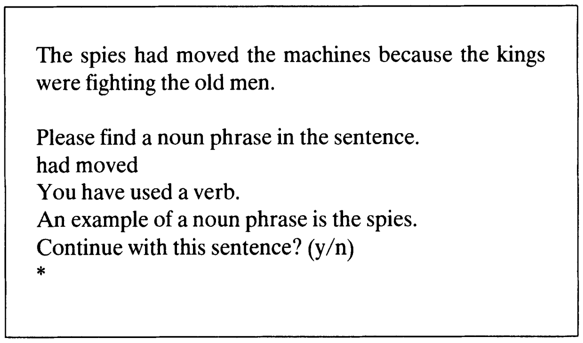
A Description

PARSER is intended to help students learn English sentence structure by giving them practice in identifying the grammatical parts of sentences it randomly generates. It is programmed in MPROLOG 2.1. (a form of Prolog) and runs on an IBM PC with at least 512k core memory.

PARSER work in the following way. A student starts up the program and ask PARSER to generate a sentence. PARSER then creates a sentence, presents it on the screen, and asks the student to identify one or another of its grammatical parts (e.g., noun, verb, prepositional phrase). If the student types in an incorrect answer, PARSER not only reports that the answer is incorrect, but also tells the student what part of the sentence has been typed in (if it is an identifiable grammatical part), and gives the student an example of a correct answer.

For example, PARSER might generate the sentence "The spies had moved the machines because the kings were fighting the old men" and ask the student to find a noun phrase in the sentence. If the student types in "had moved", then PARSER will identify this as a verb and state that "the spies" is a noun phrase which could have served as a correct answer (see Figure 5).

A student can take as much or as little time as desired to answer a question; there are no time limitations. PARSER also gives students the option of being asked more than one question about the same sentence or moving on to a new sentence after each question. A student is forced to move on only when there are no more grammatical parts to identify in the current sentence. This allows students to spend more time on sentences which seem to them to have more problematic structures.



```
The spies had moved the machines because the kings
were fighting the old men.

Please find a noun phrase in the sentence.
had moved
You have used a verb.
An example of a noun phrase is the spies.
Continue with this sentence? (y/n)
*
```

Figure 5. An example of a PARSER screen.

Generating Sentences

In order to generate sentences, PARSER uses some very basic concepts of natural language processing: phrase structure rules and semantic features. The meaning of these concepts and the method of their incorporation into PARSER will be explained below.

Phrase structure rules are rules which take the following form:

A → B

A rule of this form is understood to mean that *A* is comprised of *B*. In the study of natural languages, the *A* in the above schema is generally assumed to stand for only one symbol and the *B* for any number of them (for more details see Sells 1985: 9ff). So, for example, the fact that a noun phrase can be comprised of a determiner ("the", "a", "some", "many", etc.) and a noun can be represented in the following form:

Noun Phrase → Determiner, Noun

Rules of this kind can easily be programmed as instructions for computers. They can be understood to mean that something of type *A* can be constructed of whatever is represented by *B*. For example, the rule concerning noun phrases given above can be understood to mean that a noun phrase can be constructed of a determiner and a noun.

Phrase structure rules allow one to construct sentences through recursion, that is, by using the output of one rule or set of rules as the input to the same rule or set of rules. Below is an example of a recursive rule:

Noun Phrase → Noun Phrase, Subordinate Clause

This rule says that a noun phrase can be constructed of a noun phrase followed by a subordinate clause. In this case the output of the rule includes a noun phrase which can in turn be constructed by using the very same rule.

Recursion is a very powerful tool in that it allows many different types of sentences to be generated by relatively few phrase structure rules. However, it also has some drawbacks. In relation to PARSER, the most important one is that recursion can allow the computer to generate unacceptably long or complex sentences. (In fact, if not controlled, it can allow the generation of infinitely long sentences.) For example, the recursive rule concerning noun phrases can be used to generate the noun phrase "the cat that ate the rat that ate the mouse that ate the ant that ate the louse that ate the germ that ate the virus" in a sentence such as the one below:

The cat that ate the rat that ate the mouse that ate the ant that ate the louse that ate the germ that ate the virus was here.

The simplest way of controlling recursion is to stipulate limits on the length of generated sentences. Unfortunately, such filtering is very inefficient. Every time the program generates a sentence which surpasses the stipulated length, it must backtrack and attempt to generate another which falls within the limits. Moreover, the program is not prevented from generating another sentence which exceeds the limits on its second attempt. In addition, a stipulation on length does nothing to prevent the generation of overly complex sentences such as the following:

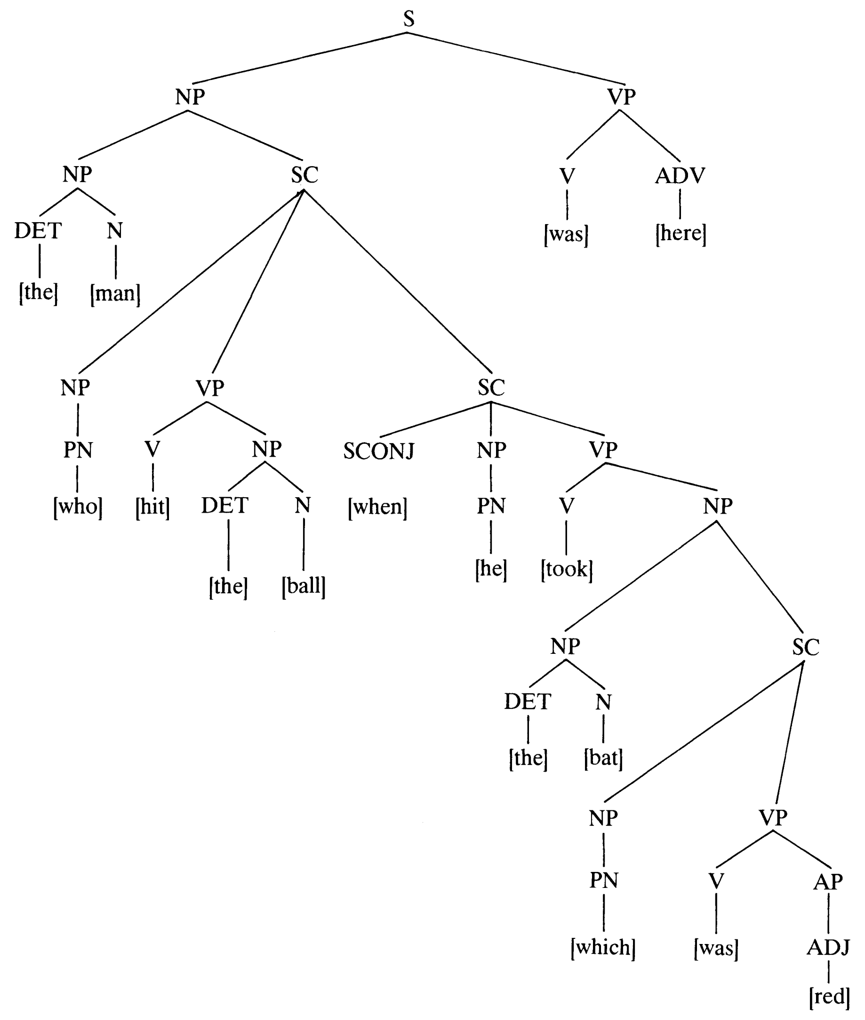
The man who hit the ball when he took the bat which was red was here.

Here length is not the problem. The sentence contains only sixteen words. However, like the sentence above it includes an extremely complex noun phrase ("the man who hit the ball when he took the bat which was red"). The complexity results from one subordinate clause containing a second subordinate clause which in turn contains a third (see Figure 6).

An additional restriction on the number or complexity of subordinate clauses would not solve that problem in an adequate manner because in certain contexts a particular level of complexity may be acceptable while in others it is not. So, for example, the noun phrase "the man who hit the ball when he took the bat which was red" is perfectly acceptable in the sentence below, although above it is not.

I saw the man who hit the ball when he took the bat which was red.

In order to control recursion, we divide all major phrase and clause types into various subtypes. Instead of saying, for instance, that a noun phrase can always be comprised of a determiner



S	— Sentence.	N	— Noun.
NP	— Noun Phrase.	DET	— Determiner.
VP	— Verb Phrase.	ADV	— Adverb.
AP	— Adjective Phrase.	ADJ	— Adjective.
SC	— Subordinate Clause.	PN	— Pronoun.
V	— Verb.	SCONJ	— Subordinate Conjunction.

Figure 6. The syntactic structure of "The man who hit the ball when he took the bat which was red was here."

and a noun ("the book"), or a smaller noun phrase and a subordinate clause ("the book which is dusty"), we say that one type of noun phrase is comprised of a determiner and a noun and another type is comprised of the first type of noun phrase and a subordinate clause. Below is a sample of the form our rules take:

Noun Phrase (1) → Determiner, Noun Bar
Noun Phrase (2) → Subordinate Clause (1)
Noun Phrase (3) → Noun Phrase (1), Subordinate Clause (1)
Noun Phrase (4) → [Who] or [Which]
Subordinate Clause (1) → Noun Phrase (4), Verb Phrase (1)
Verb Phrase (1) → Verb, Noun Phrase (1)
Noun Bar → Adjective Phrase, Noun
Adjective Phrase → Adjective

What this kind of subtyping implies is that phrase structure rules themselves are not recursive but sets of phrase structure rules are. The number of steps necessary to generate a sentence is not increased because the same set of operations is used to create all the subtypes of each type. Subtypes are distinguished from each other in that each uses a different subset of the set. For example, all noun phrase subtypes use the collection of rules which include those for noun bars, adjective phrases, and subordinate clauses. However, none use all of the rules in the collection.

The subtypes allow us to write phrase structure rules which do not produce overly complex phrases and sentences. The subtype Noun Phrase (3), for example, can generate a complex noun phrase such as "the book which is dusty" but not an unacceptable construction such as "the cat that ate the rat that ate the mouse. . . ." Moreover, we are able to decide when a complex phrase such as "the man who hit the ball when he took the bat which was red" is acceptable. We can stipulate that a noun phrase of this type can be used as the object of a simple verb phrase such as Verb Phrase (1) ("I saw the man who hit the ball. . . .") but not as the subject of a sentence ("The man who hit the ball when he took the bat which was red was here."). Subtypes thus provide us with a powerful tool for generating acceptable sentences.

Phrase structure rules allow PARSER to generate sentences which are syntactically well-formed but which, nevertheless, may be semantically anomalous. In order to produce sentences that are semantically acceptable, we use semantic features. In general terms, semantic features represent parts of the meanings of words and phrases (although they are generally not considered a suitable means of representing all of the meaning of a word or phrase). For example, the word "cat" can be said to have the features ⟨ANIMAL⟩, ⟨MAMMAL⟩, etc.; the word "book", the features ⟨INANIMATE⟩, ⟨CONCRETE⟩, etc. (see Lyons 1977: 317ff). By stipulating that words can be combined with others only when they have one or another set of features, we prevent PARSER from generating the kind of semantic ill-formedness illustrated in the sentence "Colorless green ideas sleep furiously." We say, for example, that the verb "sleep" takes as a subject only nouns which have the feature ⟨ANIMAL⟩. Since, however, we do not stipulate that "idea" has the feature ⟨ANIMAL⟩, "idea" cannot be used as the subject of "sleep".

In some cases, the subjects and objects of verbs are not in the positions where one would normally find them. The following sentences illustrate the problem:

While walking, the man saw Mary.
Mary saw the man whom Jack killed.

In the first sentence "walking" has as its subject the noun phrase "the man", even though this phrase does not occupy the normal English subject position to the left of the verb. In the second sentence "whom" is the object of "kill", even though it does not occupy the position in which objects in English are normally found. Nevertheless, semantic restrictions on "walk" and "kill" operate as though the "moved" words were in normal position.

In order to handle such cases, we have not simply multiplied the number of phrase structure rules and rules stipulating allowable semantic combinations. Instead, we have borrowed a device used in Chomskyan Government-Binding theory: the concept of empty nominal categories. An empty noun phrase functions just like any other noun phrase except that it contains no words. It is often coindexed with a non-empty noun phrase in order to indicate that it has the same referent (hence features) as the non-empty phrase (see Sells 1985: 42ff). Incorporating the empty noun phrase device, the rules PARSER uses to generate "While walking, the man saw Mary" look like this:

Independent Clause (5) → Participial Phrase, Independent Clause (1)
Independent Clause (1) → Noun Phrase (1) [i], Verb Phrase (1)
Participial Phrase → Subordinate Conjunction, Noun Phrase [i], Verb Phrase (2)
Verb Phrase (1) → Verb, Noun Phrase (5) [i]
Verb Phrase (2) → Verb (present participle)
Noun Phrase (1) → Determiner, Noun Bar
Noun Bar → Noun
Noun Phrase (5) → Proper Noun
Noun Phrase [i] → [ ]

These rules produce the structure presented in Figure 7. The noun phrase followed immediately by [i] is empty. The "i" indicates that this noun

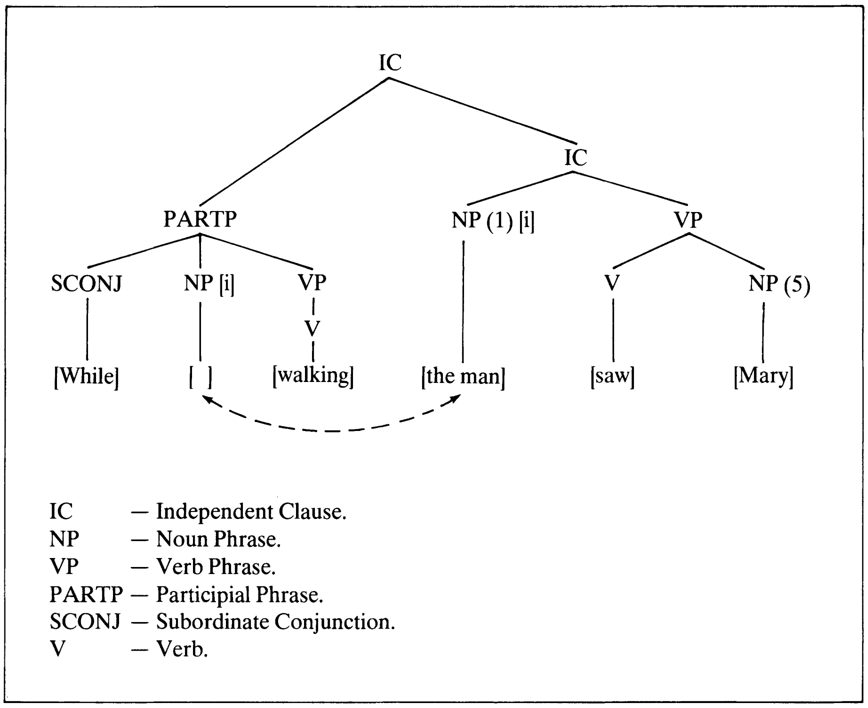


Figure 7. The syntactic structure of "While walking, the man saw Mary."

phrase is coindexed with Noun Phrase (1) [i]. Our semantic rules stipulate that if an empty noun phrase is coindexed with a non-empty noun phrase, it has the same features as the non-empty noun phrase. Since Noun Phrase (1) in the sentence under consideration is "the man", the subject of the participial phrase must have the feature ⟨HUMAN⟩. PARSER then is prevented from selecting as the participial a verb which takes only non-human subjects ("grazing", for example). Thus, by using empty noun phrases, we can employ the same rules of semantic well-formedness for sentence initial participial phrases as we do for other constructions.

A definite clause grammar notation that takes the same general form as the phrase structure rules above is offered by Prolog. However, this notation was developed to aid in writing parsers rather than in generating sentences, and would have been of only limited use to us (see Clocksin and Mellish 1984: 219). We preferred to develop our own style for implementing our phrase structure and semantic rules. An example of this style is presented below. It is the Prolog rule which generates the type of noun phrase identified above as Noun_Phrase (1).

```
noun_phrase (F,PL,NP,NT,C,1):-
        select (C,I),
        noun_bar (I,F,PL,NB,NBT,C),
        get_det (D,PL),
        append (D,NB,NP),
        add_statement_B (ans (det, [determiner],D)),
        add_matches (det),
        NT=[np,NP,[det,D],NBT],
        add_statement_B (ans (np, [noun,phrase],NP)),
        add_matches (np).
```

The rule works in a sequential manner. First, *select* (*C, I*) chooses from a list of possible *noun_bars*. Then the particular *noun_bar*(*I, F, PL, NB, NBT, C*) which has been chosen uses the semantic features encoded in *F* to obtain an appropriate noun, or adjective phrase and noun. Next *get_det*(*D, PL*) chooses a determiner ("the", "a", "some", etc.) of the appropriate plurality. The determiner (D) and the noun bar (NB) are then appended to form the noun phrase (NP) by using *append*(*D, NB, NP*).

To illustrate how this works, let us say that

PARSER already has generated the subject "the man" and the verb "hits", and is now going to create a noun phrase which will function as the object for "hits". Since "hit" can only take nouns which have the feature ⟨CONCRETE⟩, the *F* in *noun_bar* *(I,F,PL,NB,NBT,C)* will be given ⟨CONCRETE⟩ as a value. When *noun_bar* *(I,F,PL,NB,NBT,C)* then selects a noun, it only looks for nouns which have been given this feature ("ball", "desk", etc.). Let us say that it chooses the noun "balls". Since this noun is plural, the variable *PL* in *noun_bar(F,PL,NB, NBT,I)* is given "plural" as a value. This value is passed over to the *PL* in *get_det(D,PL)* then *get_det(D,PL)* can choose only a determiner which can be used with plural nouns ("the" or "some", but not "a"). Say that the determiner "the" is chosen. Then *append(D,NB,NP)* appends this determiner to the noun to form the noun phrase "the balls".

This brings us to the role of *add_matches* and *add_statement_B*. In Prolog, structures can be defined that allow information to be added and removed dynamically while the program is running. These dynamic structures allow us to generate a "parse tree" and a question list for each sentence which is generated. As each part of a sentence is generated, we use *add_statement_B* and *add_matches* to add each part ("ball","the", and "the ball" in the example) and its identification (noun, determiner, noun phrase) to the dynamic structures. The information is then used to ask questions about the sentence and to supply appropriate examples of correct answers. When the student is finished with the sentence, the information contained in the dynamic structures is deleted.

By using natural language processing techniques to generate sentences and their grammatical structures, PARSER is able to give students a practically infinite number of examples to test their ability to understand English sentence structure while at the same time giving them informed feedback. Without techniques for generating sentences and their grammatical structures, one would have to give a full description of the grammatical structure of every sentence used in the exercise. Since even a relatively simple sentence can involve as many as twenty structural specifications, the amount of work necessary for even a small pool of sentences would be considerable; to create a CAI lesson on the scale of PARSER would be practically impossible.

CONCLUSION

In this article we have tried to show how natural language processing can be an important part of CAI software. In VERBCON and PARSER, a number of natural language processing techniques are used to develop drill-and-practice courseware for English grammar instruction. These techniques allow us both to create lessons easily and to give students informative feedback.

The techniques can be applied to areas other than English grammar instruction, particularly foreign language instruction. A concrete example of this is provided by COMTEXT, a set of CAI templates for reading comprehension being developed at The University of Western Ontario by Glyn Holmes and Alan Bailin (see Holmes and Bailin, forthcoming). In COMTEXT, a deterministic parsing strategy based on the one in VERBCON is used to determine the identity of French verb forms. We hope that other CAI projects will find the techniques discussed here useful as well.

However, VERBCON AND PARSER only begin to explore the possibilities for development. We expect that in the near future far more sophisticated natural language processing techniques will be used to present students with challenging controlled exercises.

PRINT REFERENCES

Burghardt, Wolfram. "Language Authoring With 'Comet'." *Computers and the Humanities*, 18 (1984), 165—172.

Clocksin, W. F., and C. S. Mellish. *Programming in Prolog.* 2nd ed. Berlin, Heidelberg, New York, Tokyo: Springer-Verlag, 1984.

Culley, Gerald R. "Generic Or Specific: Having It Both Ways with Generative CAI." *Computers and the Humanities*, 18 (1984), 183—188.

Holmes, Glyn. "Creating CAL Courseware: Some Possibilities." *System*, 11 (1983), 21—32.

Holmes, Glyn and Alan Bailin. "COMTEXT: An Authoring System." In *CALL: Systems, Templates and Strategies.* Ed. Wm. Flint Smith and Robert Ariew. Forthcoming.

Lyons, John. *Semantics.* Vol. 1. Cambridge, London, New York, Melbourne: Cambridge University Press, 1977.

Marcus, Mitchell P. *A Theory of Syntactic Recognition For Natural Language.* Cambridge, Mass. and London: The MIT Press, 1980.

Pusack, James P. "Answer Processing and Error Correction in Foreign-Language CAI." *System*, 11 (1983), 53—64.

Sells, Peter. *Lectures on Contemporary Syntactic Theories.* Stanford, CA: Center for the Study of Language and Information, 1985.

Winograd, Terry. *Language As A Cognitive Process.* Vol. 1 (Syntax). Reading, Mass.: Addison-Wesley, 1983.

SOFTWARE REFERENCES

CLEF. Gessler Educational Software. New York, 1985—87.

Pusack, James P. DASHER: A Natural-Language Answer Processor. Diskette for Apple II. Published by CONDUIT, P.O. Box 388, Iowa City IA 52244, 1982.