ABSTRACT: The development of a successful, "intelligent" parser is essential to "intelligent" CALL programs. A modification of ATN makes possible the incorporation of case grammar analysis to judge meaningfully the use of prepositions, articles, tenses, and other parts of speech. This modified ATN makes possible the difficult task of constructing a parser which recognizes semantic and syntactic differences in an infinite number of possible sentences.

This parser works with an internally generated copy of the text so as to not compromise the original text. The parser stores characters and compares them against internal dictionaries and other characters or words. These comparisons are made via several algorithms which make distinctions between such classes as Agent/Experiencer and Path/Location. The parser is able to skip around in a sentence through the use of pointers and stripping suffixes to find roots.

KEYWORDS: Intelligent parser, augmented transition network (ATN), Basic Organizer of Syntax and Semantics (BOSS), Agent, Experiencer, Path Location, Analyzing Language (AL)

If we want to develop "intelligent" computer-assisted language learning (CALL) programs, then we need to design successful parsing programs to decode syntactic structures. We need to give students a simulated expert at reading the English sentence, so that, in turn, students can both view the reading process and learn the structures used in transmitting language. To enable students to respond effectively to a simulated expert, that program should be able to recognize student input—to offer tutorials, to advise, to tally results and responses, and to react to individual learning rates by branching. "Intelligent" CALL depends on the ability to parse a language. A computer program that recognizes and interprets syntactic elements can accept varying student responses and itself respond appropriately.

But an intelligent parser is difficult to construct. Take the three words "the old train..." What word may follow the word "train"? Most people would suggest a sentence like "the old train moved down the track." But what about changing
the word "train" to a verb form? "The old train the young" changes our expectations of a sentence structure—making "old" the subject, "train" the verb. How can we train the computer to recognize differences in words? Readers generally backtrack when confronted with the fourth word, "the," and reconstitute the meaning based on the syntax presented.

Is it that easy for a computer program? And can one backtracking solution for "the old train" problem apply universally to all possible sentences? Given an article in the expected slot of "verb," assume that the previous word is the real verb. But what about "the old train the young boy saw down the tracks was moving slowly away"? Certainly, a rule or two about sentence punctuation and verb forms will begin to construct a more useful algorithm. But for every rule so designed, another sentence possibility can exist.

I decided that I could—rather than develop ad hoc algorithms based on possible permutations of sentence structures—duplicate cognitive sentence analysis and approximate the human reading response. The Augmented Transition Network (ATN), used in some parsers for artificial intelligence work, served as a starting point. I have modified the ATN to incorporate case grammar analysis in order to judge meaningfully the use of prepositions, articles, and verb tenses (as well as other parts of speech). I used Turbo PASCAL for the program's language. (The name given to this program was originally BOSS, Basic Organizer of Syntax and Semantics, but the new working name is AL, Analyzing Language. The original program worked only on a mainframe IBM computer and now operates on an IBM PC or compatible computer with at least 256K RAM, and with two disk drives.)

Case grammar analysis proved to help solve some of the parsing problems encountered with garden path sentences as well as with ordinary ones. The plain vanilla ATN worked quite well by itself, but a further parse of the sentence, using case grammar structures to target sentence patterns, described the likelihood and existence of idiomatic prepositional phrases, noun phrases, and verb tenses. For example, the "train" sentence above could be re-parsed (in case grammar arguments) according to the main subject phrase, the main verb, and the main object phrase, because the case grammar parser would look for these main elements that form the case grammar arguments rather than stall on an ATN-generated parse (using a limited SVO algorithm). The case grammar analysis component of the whole parser moves closer to the intelligent parser that would take into account some "knowledge" of the context of the sentence. I will first discuss case grammar analysis generally, and then discuss the ATN implementation.
Case Grammar Arguments: A Neutral Parsing Method

In terms of syntax, a case grammar study involves the study of logical functions for noun phrases which legislate "arguments" inherent in the context of the sentence. For example, a woman may experience fear, but she can not own it; she may own a car, but she cannot experience the car (at least not in terms of the car being an animate being). A narrator in a novel may use a consistent form of address for his experiences, a logical argument that indicates that he possesses or wants to possess these experiences rather than experience them directly. The narrator's sentences may occur passively, with few concrete active verbs. We could then claim that the narrator does not fully appreciate or react to what happens in that fictional discourse. The narrator, in that case, is an Experiencer.

To pursue the matter briefly, the case grammar linguist looks at how a sentence predication modifies meaning. Predication consists of a verb and several kinds of noun phrases which show notions of Agency, Instrumentality, Experience, Source, Goal, Path, Location, Possession, Receiver, and Force (among other kinds). Role relations determine what function a noun phrase can play in a sentence. Normally, one cannot say meaningfully that "Lightning willingly hit the boy," because a Force has "no volition," while "I willingly hit the boy" can exist because an Agent can be responsible and willful.

Prepositions and verbs are logical determinants that guide object noun phrases. An Experiencer uses few active verbs and many noun phrases that indicate Location, as in "I was in London at the time." An Agent might well write "I made my home at London then," showing an active interest in the city as well as indicating London to be a Path along life's road. Other prepositions, such as "by," and "into," can represent other role relations—"by" can indicate a conveyance, beneficiary, or instrument; "into," a location, a path. Given a list, a central dictionary of prepositions and their common representative case meanings, I can search out and tag text according to the point of view developed. However, I wanted the computer to do this for me automatically. The solutions to this general challenge proved numerous.

The rewards, however, were worth the effort: some of the idiomatic expressions used in English can be captured with case grammar rules and constraints. Whatever the context for the discourse, the parser tracks the spatial relationships for each sentence and indicates the probable structures used idiomatically. Hence, a neutral mapping of the text takes place with AL at a higher level of "understanding"—moving from simple pattern matching to
retrieving meaningful relationships among the subjects, verbs, and objects of individual sentences—than is possible with an ordinary ATN.

**Parsing: The Main Objectives of Case Grammar Analysis**

In terms of case grammar analysis, for instance, "he went under the table" is different from "under the table was a broom," because the first indicates a Path, the second a Location. A passive verb construction indicates the likelihood of a Location for the object noun phrase, and an active verb shows the occurrence of a Path. Granted, "he lay under the table" presents some difficulty in interpretation. On the surface, the case grammar/syntactic parser would list "under the table" as a Path, although "he lay" (or "it lay") might be better described as a passive state, making the "table" a Location. The context of the sentence would determine the argument. An impersonal pronoun, "it," for example, when defined as an inanimate subject, would indicate that the object noun phrase is a Location rather than a Path. Personal pronouns, on the other hand, would be tagged as Path-potentialized; animate things seem Path-oriented, even if activity ceases. The context of the sentence (in terms of the subject’s potential state—as animate or inanimate) would then give the program the proper heuristic rule for inanimate objects and animate beings. Figure 1 presents a flow chart of the problem and this solution with the preposition "under":

```
"Parser >>>>>>> "Subject" + "Verb" + "Object"

Noun Phrase

If "under"-----------------------------------------

Examine Subject---------------------Pronoun references

Animate? Inanimate?

Is Verb Passive? ("to be" verb in dictionary)

No Yes

Declare Noun Phrase a Path Declare Noun Phrase a Location

Declare Subject an Agent Declare Subject an Experiencer
```

**Figure 1**

If the subject is inanimate, then the algorithm declares the object noun phrase to be a location. If the subject is animate, then the algorithm insists on
checking on the verb—if passive, then the object noun phrase is a location, but if it is active, then the object noun phrase is a path. The subject of the sentence is then tagged appropriately.

"Volition" becomes defined by this process. An Experiencer lets things happen to him/her, while an Agent actively participates: active verbs usually indicate a logical argument of Agent for the subject, passive verbs indicate an Experiencer.

AL analyzes each preposition according to its specific spatial orientation. "In," for example, follows a similar heuristic model to "under" when parsed. If no preposition occurs, as in "I walked home," the parser flags the sentence so that "home" can be labeled as a Goal; some locational noun phrases do not require the usual "to" as a prepositional marker to indicate the Goal-state of the noun phrase. However, in the presence of a passive verb, of a "to" noun phrase as object, of a personal pronoun or formal name, of a gerund or participial, and in the absence of a Goal-centered sentence, the logical argument for the "to" marker may be an Experiencer rather than a Goal: "The rose was pleasing to John," or "that was interesting to her." The preposition "to" as with "under," demands its own set of heuristic paths, even when, oddly enough, the word is absent—though implied and possible if added artificially, as in "I gave to her a passing grade."

Case grammar analysis and parsing procedures work cooperatively. When the parser locates noun phrases, case grammar analysis then renames these noun phrases with logical determinants of meaning. Case grammar studies show the actions and state changes in a text across several sentences, particularly due to preposition markers, after the ATN parsing system tags main sentence elements according to general syntactic rules.

What AL Must Know: Initializing Variables and Dictionaries.

The computer itself must be given physical limits. Specifically, what are the essential words for a syntactic parser? What, indeed, is a word? And, more importantly, how does the machine read natural language input? Also, what are the "tricks" used to make sense of language's fluid state? (After all, there are many, many possible sentences; a parser must somehow be able to decipher each possibility.)

A word, by itself, to the computer, is nothing more than a series of characters bounded by blanks on either side. The machine cannot read except by comparing a list of words in some kind of dictionary against the word being analyzed. In general terms, the computer checks each character for a template match until it succeeds or fails. In AL, a set of function words—words that often
indicate the kind of subject, the kind of direction to the sentence, and the kind of sentence structure being used—were used as pivot points for the parsing procedures. These words are common articles, pronouns, conjunctive adverbs, and conjunctions. I have also included the common verbs, nouns, and adjectives for other parsing structures; verbs, particularly, have had to be extensively defined in terms of number, voice, and mood.

Words (especially verbs) exist in pre-assigned frames, so that the lexical functions can also be determined at the same time the parser traverses a sentence; these functions are organized by the slots available to them in terms of general idiomatic use. Nouns can be plural or singular; verbs can be tagged according to tense type, voice, mood, person, and number; prepositions can carry various kinds of case grammar arguments. This dictionary-frame carries by initial default an empty set of values until filled with significant ones.

AL also keeps track of the case grammar relations being developed and of the object and subject noun phrases being processed, so that the verb used can be checked for its consistency with the noun phrases in the sentence. For example, intransitive verb forms are contained in one dictionary—the root forms—and, if the word search for an intransitive verb is successful, the conditions for an intransitive-verb sentence are assigned for that current sentence (and for that verb form) and then compared to the arguments of the accompanying noun phrases.

AL assumes several things about a machine-readable text. If AL finds a mark of closure—period, quotation mark, parenthesis, exclamation mark—AL looks ahead for the next character or blank in the line of text. If the next character is another boundary marker, then the same procedure as described is used again—looking ahead one character or blank. In some cases the previous word is tested to make sure that it is not an abbreviation; in AL even ellipses are found internal to sentences. Should the program find several blanks in a row, then AL assumes that this is the end of a sentence.

There are, as well, important initialize procedures: commands that define the set of upper and lower case characters that are used in English and the kinds of characters that can be used for punctuation and commands that locate the dictionaries used (nouns, verbs, adjectives, and other significant words) and the text that the user wants analyzed. A number of static (fixed set) dictionaries of function words, such as "the," "but," and "however," are also initialized. Other static dictionaries include conjunctions, pronouns (relative, demonstrative, interrogative, indefinite, personal, possessive, reflexive), articles, prepositions, subordinating conjunctions, negative words ("no," "won't," "never," "can't," and
others), and conjunctive adverbs. Adjectives, nouns, verbs—meaningful words—are in user-definable dictionaries that allow addition or revision (accessible to any word processor). [AL allows user-defined dictionaries to be developed, with only a few restrictions, the chief ones being that the words appear in uppercase, that blank spaces appear after each entry, and that each entry appears on a separate line in an ASCII text file.]

Physical limits had to be placed on AL: punctuation marks and what they mean; the length of the line to be used safely as a buffer (the length of the computer terminal screen line); how many characters are assumed to be the maximum for any one word; the number of words that can exist in the dictionaries without taxing the memory system; the form of the output and the input databases, and where to locate them for the computer's use; and the kinds of constants, variables, and other types of essential "words" necessary for AL.

For example, the parenthesis, the exclamation mark, the period, the semicolon, the comma, and the rest of the punctuation marks are defined. The line length can only be 81 characters long; the word length for a static word entry in the internal dictionary can only be 14 characters long. The line length limit is more or less determined by the limits of the terminal screen and the computer's definition of an acceptable line length. The word length for a static word entry is arbitrary but the development of a utility program that allows unlimited word length would be too costly in terms of the computer's memory limits. [With 1,000 entries, an additional character to the word length adds 1,000 more characters to be stored; in terms of the computer's storage capabilities, this isn't much, but word frequency tables suggest that word lengths over 14 characters are rare enough so that saving time and computer memory is worth the limitation.]

Another limitation is how AL treats the input text. AL locates a line of text from the file defined by the user as the input text. The computer looks through disk's contents for the named file, and, if there, latches onto it. It then records this line into a temporary storage area called a buffer, a buffer that has been defined in my program as equivalent to an array of spaces 81 across. This buffer line itself will be placed in another buffer, one that may grow as the text being analyzed is read line by line. The analysis of one sentence depends on keeping these buffer lines intact and yet distinct in terms of line lengths and word lengths. The program manipulates the characters of a text in many ways, making it important to leave the original text untouched.
The buffer line, to be sure, required a way to keep track automatically of the characters in the sentence without having to guide the tracking activity character by character. The pointer variable allows this. An initial step within the analysis procedure marks or checks off the beginning of the sentence—without reaching the end of the sentence yet—making sure that a number of variables are set to the beginning of the line. These variables are called pointer variables, and are, at first, instructed to point at the first character of the text; these pointer variables, depending on the need for them, will jump around from character to character in the line of text. Pointers will also be created for individual words, as well as case grammar arguments and clauses. Pointer variables are dynamic in that they can be re-used, changed, or even point at each other, without really carrying along those items (a word, say) around, at the same time that they can create themselves when needed and they can vanish when no longer required. Pointer variables just address things; they do not contain them. Pointer variables allow the controlled parsing that ensues.

AL uses some of the pointer variables to "run" over the sentence until it locates what has been determined to be a word. When AL finds a word, it then sequentially tests the word against a number of dictionaries. Any time that a word in one dictionary matches the word being analyzed, the word is recorded according to its class and its potentials are marked. For example, a past tense transitive verb receives the appropriate pointer variables indicating those conditions. Pointer variables and some integer variables then keep track of the place in the sentence, the words in the sentence, and the pathways that maybe taken, (a known word can shift the word search, for example). The programmer releases control of this tracking activity to these pointer variables—for they keep track of themselves. These pointer variables reflect language nicely.

As another limitation, AL converts each buffered word into uppercase characters, because the central dictionaries are in uppercase. Otherwise, the dictionaries would have to appear in both uppercase and lowercase character sets. The text itself, of course, remains inviolate, since AL uses only internally generated buffered lines and words.

Every word also undergoes a stripping procedure that analyzes the ending, the suffix, to see if the suffix can be removed, and to determine if the root word belongs in the syntax or semantic dictionary. The characters "s," "ly," "ing," and "ed" are removed from the word, if found, and the resulting word compared to the dictionary's database. Though the original list is about 1,000 words long,
these suffix removal procedures extend the actual number of words in the
dictionary to many more possibilities (including gerunds, plurals, and so forth).

**Parsing: Inside AL's Augmented Transition Network**

After being told what words are (a set of characters with blanks on either
side), and what sentence units are composed of (marked by punctuation), AL
isolates a string, a set of words within one sentence unit, and goes to work. Any
output appears when the parse succeeds; intervening steps are erased, being part
of a *scratch pad* memory system. The sentence-level parser (the first parse of the
sentence) is based on an augmented transition network (ATN) described in
artificial intelligence research.¹

The ATN component of AL first hunts for a subject of a sentence, defined
as a noun phrase with a determiner, a modifier, and a *head* word (the pattern
need not contain a determiner or a modifier, but it exists as a universal template
for the computer). AL checks the first word of the sentence to see if it is a
determiner (an article, perhaps), matching it with dictionary-defined words. If it
is a determiner, then AL can assign a *flag*, a marker to the current word that will
indicate the part of speech as an article or as a determiner, and move on to the
next step. There AL checks a limited bank of words in the dictionary for
adjectives most commonly used in English. It may then look for a noun, unless it
finds a second adjective, at which point it continues to search for adjectives, until
it has found a noun or a verb (completing the noun phrase). At each successful
match, a flag is set, indicating what kind of word the parser has found. For the
assignments being made, and for a graphic display of these activities, see figure
2. The term *arc* is the direction or the current request for the analysis, and *action*
is the possible assignment. Arcs can re-direct the search (i.e., skip around), based
on the slot assignments being made. Arc (3), for example, calls up the general set
of search routines for a noun phrase, used earlier (perhaps) by arc (1).

If AL succeeds so far, it will label perhaps the first three words a noun
phrase, assuming that a determiner, an adjective, and a noun are there—it may
just find the first word to be a noun, hence calling the first word a noun phrase
by itself. Then the program looks for the verb in the next word batch, again
searching the global dictionary for "to be" verbs and common verbals.
Depending on the kind of verb it locates—transitive, intransitive, or simple—the
augmented transition network parser moves through heuristic pathways,
**Figure 2:** Simplified Augmented Transition Network (ATN) for computer sentence parsing.
looking for syntax clues such as the word "to" or other noun phrase structures, and jumping when required into other pathways. All conditions for the noun and verb transition networks must be met before the parser begins to look for the object of the sentence.

At some point, if the sentence contains only three basic parts, then the entire string (labelled text) is sent along to a text formatting procedure, and the syntactic information is stored temporarily for the case grammar analyzer. With patterns using relative pronouns, the augmented transition network parser first locates a relative pronoun, and checks a scratch-pad memory for any hold messages that indicate a previously stored noun phrase. (A hold message is quite similar to the telephonic hold: the computer continues the search only after completing another; if on hold, AL will assume that the true subject of the sentence has yet to be found, assuming that the parser has found a complement clause or similar syntactic pattern, just as the caller on the phone must wait for the person to return to the caller's business.)

A noun phrase might be repeated, continuing a complement-embedded sentence. It may claim that the noun phrase before the relative pronoun is a potential subject for the following embedded Subject/Verb/Object (or SVO) pattern clause. If the hold box contains a noun phrase, then the relative pronoun is bypassed, in a sense, in order to label the clause properly.

Verbs and objects, in turn are assigned within the structure of the sentence. At some point the embedded SVO pattern or phrase may be given a modifier status, and the parser moves on. The hold location is erased whenever a relative pronoun shows that a noun phrase has been found, allowing the parser to perceive a multiple set of references and embedded clauses. Complement clauses, noun phrases, absolute phrases (depending on the comma, the placement of noun phrases, and SVO patterns), and verbals become defined and labelled. Depending on the sentence, the hold address serves many analyses within one sentence. The web of heuristic procedures rarely works the same way twice, simply because sentences often differ.

When the sentence in the buffer has been tagged in this simple way of labelling the words that exist according to the central dictionaries, the parser begins anew with yet another set of procedures.

AL again starts at the beginning of the sentence, traverses the sentence as labelled, looking for words that have not been identified. Some words can be misidentified in the first pass. The word "that," for example, can be tagged a number of ways; AL might label it a demonstrative pronoun. AL examines the
words on either side of this demonstrative pronoun to see what kind of word the demonstrative pronoun precedes or follows. With the word "that," the basic algorithm suggests that if the following word is an article, another demonstrative pronoun, a personal pronoun, or an indefinite pronoun, then the parser should re-label the "that" in question as a relative pronoun. Other grammatical re-labelling procedures inside AL include an infinitive tester, an adjective tester, a noun tester, and a particle locator.

Most of this information remains internal to the workings of AL, since the final output desired for each sentence could not be displayed without taxing the patience of anyone reading the computer-generated analysis, if the full array of heuristic path-ways were shown. Inside AL the vestiges of some diagnostic procedures could be revived easily enough to generate a one or two sentence analysis in great detail, but the prototype working version does not have those readily available to the ordinary user. I plan to modify AL so that the ATN will show the paths taken during the sentence analysis with a graphic display of the choices being made, if the user wants that information (though AL will then only exist on a 640K machine, because of the demands on RAM).

**Parsing: Inside the Case Grammar Parser**

AL, in the next stage of sentence analysis, runs over the sentence again to begin looking for case grammar arguments. AL pauses and displays the ATN analysis by printing out the original sentence as buffered and stored, along with a syntactic map of the sentence as parsed by AL's ATN.

AL, in the case grammar analysis phase, examines the content of the sentence in terms of the verb, object, and subject. It develops a network of case grammar arguments for the word classes that exist as found by earlier procedures. If the verb has been defined—and if the subject has been determined by the pronoun choice—any use of prepositions can now be determined according to case grammar arguments. Each preposition has its own set of heuristic pathways, as indicated earlier. The verb, the kind of subject, and the kind of preposition interact inside AL to re-label the parts of the sentence to indicate general patterns of case grammar arguments.

Some thirteen case grammar arguments can be tagged with limited success: agent, patient, instrument, experiencer, source, goal, path, location, possessor, force, conveyance, beneficiary, and duration. The results obtained with force, beneficiary, duration, and conveyance procedures are not satisfying, and the actual data analysis has been limited to agent, experiencer, path, location, and source, at
least in the current version. Much remains to be done with this set of heuristic algorithms: AL tends to be accurate about eighty percent of the time in terms of the case grammar arguments agent, experiencer, path, and location, and proved adequate for the kind of narrative analysis originally planned.

Let us look at a sample algorithm from the case grammar analysis. After the sentence analyzer has begun to parse successfully the kinds of words in a structure, after some words seem well defined, and after case grammar arguments are beginning to fall into place, this algorithm exists:

```
If (Subject = True) Then
  Begin
    If (Transverb = True) Then
      Begin
        Runner@.Wrdpt@.Argpt := Agent;
      End;
    Else
      If (Intransverb = True) Then
        Begin
          Runner@.Wrdpt@.Argpt := Experiencer;
        End;
      End;
  End;
```

Figure 3

The odd looking string of connected characters `Runner@.Wrdpt@.Argpt` is a pointer that points at another pointer that defines the case grammar argument of the current word. Depending on the local condition of the verb, either transitive or intransitive (defined for PASCAL by myself as boolean conditions `Transverb` and `Intransverb`), this fragment of code may label the current word as an Agent or an Experiencer. The operant boolean conditions have been defined elsewhere in the program according to other conditions inside of other procedures. The fragment will work only if `Subject` is true (here, a personal pronoun has been found in the sentence prior to using the above algorithm) and only if the program's heuristics now call for it.

In the mainframe version of AL (the old BOSS), after the text has been parsed, counters tallied, and some diagnostic messages printed beside and below the re-typed text, such as the syntactic map to the side and the messages that indicate the major case grammar argument for the sentence (centering on the narrative pronoun), then at last the computer prints out the final count of the
word classes used, presents graphically the sentence lengths, delivers a final overall judgement, and counts the number and type of case grammar arguments found in the entire text. AL then signs off, reminding the user that any analysis governed by the computer remains fallible given the condition of the programmer's linguistic strengths and weaknesses. [With separate algorithms for each preposition and with many interlinked algorithms for differing sentence constraints on case grammar arguments, AL will need more space than it seems likely to find at this point to describe the full set of solutions used for case grammar analysis. I'll need to write the book.]

Applications
This level of understanding a sentence helps to unlayer the idiomatic use of prepositions, articles, and verbs. Given a fairly schematic and rule-driven process of deciphering the role relationships in a sentence, most idiomatically correct sentences (that do not depend on separate discourse environments, like baseball jargon) can be generated or read. Some very limited comprehension can be claimed for a computer program that will read the case grammar arguments in a text. The message read will, of course, be quite neutral, dependent on spatial directions and role relations inherent in and admissable to a sentence. The overall aim, however, is retained: typical structures and usable frames of reference are displayed as a set of rules for the linguist (and for the computer programmer). Also, as a side benefit, garden path sentences tend to lose their ambiguity, assuming that case grammar analysis can be incorporated into the computer program with some completeness. [Only a fully operational syntactic parser that can function with varying discourse environments will be able to decipher all idiomatic sentence structures. AL makes no such attempt in its current design. AL has been able to operate on job application letters in technical writing and business communication classes, but still requires an instructor's hand for matters of content.

At present, the program AL exists as a diagnostic tool. It reads and describes the sentences found in a text. Further work would include making AL interactive, so that typed sentences could be parsed, analyzed, and placed into some context, AL reads, but it does not yet respond. The frame needs a house in order for AL to be termed "courseware." This program remains an ongoing project aimed toward that direction. [As of this writing, there is hope that I can incorporate AL into an expert system that will respond to various writing contexts. The size of AL will, of course, increase. As in most expert systems, the
rules followed will be displayed, for the writing context and in the sentence structures allowed.]

Courseware that can reflect the parsing process in its own response and in its own design begins to sound like the "intelligent" machine we have long waited for. But why do we need one? Simply stated, the flexibility gained by that approach outweighs the complexity of building an intelligent parser. We would not replace the teacher in the classroom, but we would allow the class the practice needed to learn a language through a program that resembles the so-called "normal" conversation in a classroom, a controlled environment that addresses a context-sensitive discourse model, perhaps with each class session.

The intelligent parser would then be the basis for lessons depending on differing discourse environments in individual classrooms. One would still need the same sentence structures, whether the lesson involves the sales letter or the "bad news" letter in any business communications course—or the laboratory report or the feasibility study in a technical writing course. The intelligent parser could then reinforce the learning process by being both reproducible for many contexts and flexible for many levels of growth. Students could learn at their own rates the sentence structures necessary to understand any given discourse environment. And the instructor could then use that courseware to move effectively through a number of assigned areas in the intensified learning process (assuming that students use this courseware in the place of homework or workbook assignments on their own). The full parsing system that I am working with is a step in that direction. [As indicated earlier, I am now developing a rule-based expert system, one that will operate as the front end to AL, and one that will respond according to a number of discourse-specific environments.]

Endnotes


Author’s Biodata

Dr. Harold Hellwig teaches advanced writing courses (business and technical writing) at Iowa State University in the Department of English. His
research at the University of California, Los Angeles, where he received his Ph.D. in English, centered on the use of computational linguistics in literary criticism.

Author's Address

Dr. Harold Hellwig
Department of English
Iowa State University
Ames, IA 50011