DEVELOPMENT OF AN AUTOMATIC ENGLISH GRAMMAR DEBUGGER FOR CHINESE STUDENTS: A PROGRESS REPORT[1]

Hsien-Chin Liou, Hui-Li Hsu, Yong-Chang Huang, and Von-Wun Soo

National Tsing Hua University

Abstract

This paper reports the research about development of an automatic English grammar debugger (on a personal computer) for Chinese college students, based on an analysis of the errors they made in their compositions. The first stage of development is devoted to an error analysis of 125 writing samples and classification of the errors into 14 main types and 93 subtypes. To implement the grammar debugger we first built a small dictionary with 1402 word stems and necessary features, as well as a suffix processor which accommodates morpho-syntactic variants of each word stem. We then built up an ATN parser, which is equipped with correct phrase structure rules and error patterns. In addition, a set of disambiguating rules for multiple word categories was designed to eliminate the unlikely categories to increase the precision power of the parser. The current implementation enables detection of seven types of error for a text input and response of corresponding diagnostic feedback messages. Future research will be focused on refining the grammar debugger to detect more types of mistakes with more precision and on providing appropriate feedback messages to diagnose students' deficiency as well as operations for students to edit their errors.

Introduction

One of the reasons which make English writing classes formidable for language teachers in Taiwan is the seemingly endless correction task of many learners' grammatical mistakes such as subject-verb agreement and article usage. This experience motivated our

[1] The research project is sponsored by National Science Council (#NSC80-0301-H007-15) in Taiwan, Republic of China. We would like to acknowledge our research assistant's (Kuei-Ping Hsu) full-time dedication to the project.

use of computer software to alleviate teachers' burden. If a computer program can help detect or even correct grammatical mistakes in students' compositions, it will reduce the tiring part of revision process and leave more time for human teachers to work on higher-level re-writing tasks such as revision of contents, organization, or expressions.

For this purpose, we have tested to what extent a commercial software package, *Grammatik IV* [1] could help our students. It was found that only 14 percent (10 out of 70) of all the categories that *Grammatik IV* had detected are substantive grammatical errors which writing teachers are serious about. What is worse, the package misses some of the significant errors frequently made by students due to the package's limited capacity, and generates false positives and misleading messages such as those in the following brackets.

(1) *Having listening _ the teachers' word, I was not surprised at the poor score I got as I didn't do the question with caution.* [Passive voice: 'was surprised' Consider revising using active]

(2) *There <u>were</u> great man in the world whom I respected forever.* [The context of 'whom' indicates you may need to use 'who']

(3) *These occupy successively lower vanges on the scale of computer translation ambition.* [Usually 'these' should be followed by a plural noun.]

The failure in *Grammatik IV* is due to erroneous analysis of sentence structures (as in all the three above) or rigid conformity to rhetorical conventions (as in (1) and (2)). The disappointment with *Grammatik IV* motivated our research on the development of an automatic English grammar debugger which can detect the major mistakes unique in our students' compositions. Concurrent efforts such as Chen and Xu [2] have been initiated, which, as complementary to the present research, has much to be desired regarding their global design and achievements. For example, the error types their debugger handles are not based on corpus but on the researchers' intuition.

The Research

This project proceeds in two stages. Stage I is devoted to analysis of errors in students' compositions, categorization of error types, and formulation of computer processable rule patterns based on the categorization. Stage II concentrates on implementation of the grammar debugger on a personal computer.

Stage I Error Analysis and Categorization

We have collected over 1000 hand-written compositions, the corpus of this project, written by our students mainly with engineering backgrounds. The average length of the essays is about 200 words. To facilitate future testing of the debugger, we have keyed in some 194 essays (hereafter referred to as the sample database); the rest of them will be keyed in by the time the project is finished. A textual analysis mainly for syntactic mistakes has been conducted on 125 essays from the corpus. We have found 1659 errors and used a database package, dBASE III Plus to manage the errors. We then classified the mistakes into 14 major types and 93 subtypes for all the data we have analyzed. The rest of the corpus will be analyzed by the end of this project to update or refine the categorization. To measure the gravity of the error types, we adopted two criteria: frequency of occurrence and levels of hindering comprehensibility. Frequency of occurrence is measured by dividing the number of a certain error type by 1659, the total number of errors. The results, descending distribution of frequency in both raw numbers and proportion for major types and subtypes, are shown in Appendix A. To obtain a measure for the second criterion, level of hindering comprehensibility, we asked two native speakers (associate professors in linguistics) to grade examples taken from each subtype on a scale of one to four (meaning *bad* to *very bad* for comprehension). Results from the two criteria were used to screen all the error types. Lastly, we chose those categories which had higher frequency, were perceived worse and more easily processed by the computer -- under mainly a syntactic approach, before we formulated the categories into rule patterns.

To make the error types processable by a computer, we have tried to formulate error patterns or represented the errors as explicitly as possible so that computer programs may recognize/detect them. Here, we use the subtypes under Verbs as examples to illustrate how error patterns and pattern matching rules were formulated. All the subtypes are listed in Table 1.

Table 1

Subtypes of Errors Under the Verb Category

V1 (be V; redundant be verb; double finite verbs)	*People could contact with friends when they were lived away.*
V2 (modal + past verb)	*If you use it carefully, it could made many work for you.*
V-sub (verb subcategorization errors)	*They try their best to stop them happen_ again.*
VT (wrong tense/aspect)	*If the war happened, we can never live a good life.*
VT-1 (verb tense disagreement between clauses)	*If we were not interested in the basic research, then we will not go ahead any more.*
VT-2 (tense disagreement in a compound)	*...we must avoid hazardous by-product of science and utilized the good points of science.*
VT-3 (tense disagreement at discourse level)	*On holidays, I often went out of Taipei. I usually ride my motorcycle enjoying the speed of wind.*
VT-4 (contracted form fails to show plural form)	*It's rainy last weekend.*
VF (wrong verb forms -- passive/progressive forms)	*The classmates and the teacher are all keep in my mind.*

We then pulled out all context fields of each error type from our database and examined how the errors were manifested. For instance, all the contexts of V1 errors are listed in Table 2.

Table 2

All Contexts of V1 Errors

Record#	context
29	... people could contract with their friends and daily when they were lived away.
68	... then many dangerous thing will be happened.
506	Scientists have done a lot of works which made our living pattern is different from those days.
639	... the earth would be die at last.
692	... although they are not necessary improve our material life directly.
782	Because the scient is progress too fast.
817	It is seem great for the results coming out from science.
833	Although science makes our lives more comfortable, is it all do good to us?
885	Science has occupied a part of our life, and we are enjoy the development and achievement that science bring to us.
911	... I was fortunately passed the entrance examination ....
964	All of them made the earth never be suitable to be lived.
1040	All my life was began to be contained in the textbooks....

Here we take three error types, V1, V2, and V-sub as examples to illustrate how error patterns were formulated, or what we designed as a solution if the pattern could not be represented in a formal way. First, the **V1** error pattern can be described as a be verb plus another non-be verb, which has a feature of [intransitive], or [transitive] followed by a noun phrase at the verb phrase level. The only exception is the error in Record number 506 which requires another pattern to describe: causative verb, make plus finite verb be. More explicit rules can be written as follows:

```
a'  V[b]  X   - V[vi]
               |_ V[vt]  NP
b'  V[c]  X  V[b]
```

(V[b]: be verbs; X: wildcard symbol; V[vi]: intransitive verbs; V[vt]: transitive verbs; NP: noun phrase; V[c]: causative verbs)
(Note: Tentatively X is defined as an arbitrary number of words.)

Likewise, the error pattern for **V2** can be described as:

modal V-ed/V-en
(read as a modal such as should, could followed by the past tense or past participial form of a verb).

V-sub concerns problems with verb subcategorization. Referring to categorization in the framework of generalized phrase structure grammar [3], we have classified verbs into 33 categories (see Appendix B for detail). Since we have found that it is impossible to formulate error patterns for the V-sub type, we have attempted to represent the correct patterns instead. The correct representation enables mapping of verb patterns of the erroneous input onto the correct representation. As we have not completed this part, it will not be discussed in the remaining part of this paper.

Stage II On-line Implementation

Before we explain the work in this stage, we would make a note: the current project does not deal with misspellings that spelling checkers in commercial word processing packages have achieved to a very satisfying extent. For this project, the implementation work is divided into three phases and programmed in C language. Phase one concerns preparation of a small machine readable dictionary. Phase two involves construction of the electronic grammar debugger itself. Phase three pertains to phrasing and delivery of feedback messages.

Phase I. A survey of literature indicates that there are several comprehensive machine readable dictionaries available such as Longman Dictionary of Contemporary English, Webster's Seventh Collegiate Dictionary, Collins Bilingual Dictionary, and Collins

Thesaurus [4, 5, 6]. As our learners have limited English vocabulary and the project is exploratory in nature, we decided to make a small dictionary on our own to meet the immediate needs. Our experiences with this small dictionary, however, will help selection of the crucial information and access methods when we adopt an electronic comprehensive dictionary in the future. For our own dictionary, first, a program was written to extract word types from our sample database and formed the core of our dictionary entries. There are currently 1402 entries in the dictionary. Proper nouns like Chang, Tsing Hua are tentatively listed in the dictionary alphabetically. Each of the words is attached with (a) its part-of-speech information and (b) necessary features. Note that we have selected only the more likely part-of-speech information which our learners use in their English writing; we have not encoded rare usage in our dictionary. Ideally we hope the selection and ordering of word categories reflect the frequency of occurrence of each word, yet this requires further research. This selective approach has the disadvantage of encountering more unknown words if a learner's essay happens to be of higher quality. However, the reason why we adopted the simplification strategy is to save the dictionary space and increase the parser's precision. A sample of the dictionary entries and their affiliated features is shown in Table 3.

Table 3

A Sample of Word Entries and Their Selected Features in the Dictionary

Noun:	count/noncount; vowel/consonant in the initial phoneme (V/C)
Adjective:	single/multiple syllable (S/M); V/C
Adverb:	subcategories (8 classes); S/M; V/C
Verb:	subcategories (33 classes)
Pronoun:	singular/plural/both (S/P/B); person (1st, 2nd, 3rd); case (subject/object/possessive)
Determiner:	S/P/B

The entries in our dictionary are mainly stems of words, or headwords. To accommodate suffix changes of word stems, we have designed a suffix processor as suggested in the EPISTLE text critiquing system [7] by adopting the concept called a distributional lexicon [8]. The processor is equipped with information about (a) rules of changes concerning word categories (e.g. from verb to noun) or the inflectional features (e.g. from plural noun to singular noun), and (b) associated actions (e.g. omitting -s can reform a noun stem). By means of a search procedure to correlate rules and suffix changes between the variants and headwords, the suffix processor ensures that the dictionary can identify the following three types of morpho-syntactic variants of each corresponding headword built in the dictionary: (a) the inflectional suffixes such as -ing, -ed, -s (for both verbs and nouns), (b) the derivational suffixes such as -ly in happily (from happy), -ful in cheerful (from cheer), and (c) markers of comparative and superlative degrees, -er, -est (such as hotter, or fastest). In this way, our dictionary can cope with natural English texts without building all the derivations as respective entries in our dictionary. To increase the processing efficiency, we grouped the rules above so that when a word like getting is encountered, it is assigned to the -ing group. This can save the searching time among all the suffix rules.

To cope with irregular forms of verbs, we have designed a table which lists the root form, and irregular changes of verbs. If an irregular verb like began is found in this table, it is associated with the feature past tense for later processing and its root form begin. Then, the processing directs to our dictionary and attaches affiliated features of begin to began.

In addition, we plan to build up a phrase dictionary and a dictionary of common problematic words to cope with errors in, for instance, sentences (4) and (5).

(4) *The misuse of the science results to the terrible thing of the rest part of the earth.* (should be results in)

(5) *We know that science is effected to human life seriously.* (should be science affects human life seriously)

Whether these dictionaries are to be integrated into a parsing process (to be described shortly) or remain individual processors is to be explored, though Stock [9] suggests the former being more profitable in their system.

Phase II. In phase two, a parser was built and augmented by pattern matching of error types in order to automatically detect grammatical mistakes in a text input. Most of the work has been completed, whereas the other has been planned or under way.

The error patterns obtained from the analysis in the above section were tentatively classified into eight levels of processing, based on ease of manipulation by the computer or linguistic analysis, if applicable. The classification will be revised as we analyze more students' essays, generalize more and finer error patterns, and encounter bottlenecks.

(I) matching strings: For instance, the mistake in (6) can be easily detected when we simply search for the words 'Although/Though' and 'but'.

(6) Although my high school years were full of pressure, but I still found my ways to relax myself.

(II) matching strings and sets: For instance, the mistake in (7) can be detected when we search for the words No matter and a set of question words such as when, where, who.

(7) No matter eating, clothing, living, and walking, we rely on science.

(III) using the suffix processor to cope with errors related to a certain category of words: The technique can, for example, handle the problem of pluralizing uncountable nouns. After failing to match the word informations as in (8) in our dictionary, the suffix processor (designed to extract a possible stem, or root form for a word) can be used to reform the stem information. Since the countability feature for information indicates that it is uncountable, we can detect the nature of its error: an uncountable noun should not have a plural form.

(8) We must depend on some instruments like radio, computer to receive informations.

(IV) incorporating information in the dictionary into string matching: For instance, the mistake in (9) can be detected by matching the word more and searching for part-of-speech information of the following word in the dictionary. During the latter process, the suffix processor is activated to attach the feature [simple] or [comparative] degree to the word. This corresponds to the error pattern, 'more' + comparative degree of adjective/adverb, and the debugger can flag this mistake.

(9) The weather becomes more hotter than before.

(V) looking the problem up in a dictionary for common problematic words or phrases: As mentioned before, some of the students' mistakes are related to a specific word or phrase. This phenomenon will lead to construction of a specific dictionary with the hope of detecting such types of errors more effectively. In addition to problematic words, resolution techniques for detection will be built in the dictionary. This approach may help solve some of semantic problems which are not very meaning-dependent such as (10). With the help of parsing, the program can detect the mistake: misuse of an adjective for an adverb. With the special dictionary, the program enables specific diagnosis of a common error type, confusion between everyday and every day (because of very similar forms).

(10) A lot of people feel nervous everyday.

(VI) using syntactic parsing and pattern matching: This level will be explained in more detail shortly as it is the main mechanism by which the most of the implementation work has been accomplished.

(VII) using semantic processing: Most of the diction problems fall into this category. This will be a very challenging problem as the information conveyed in the essays of our corpus is not within a limited domain. We have not yet had a clear idea of how to cope with such problems.

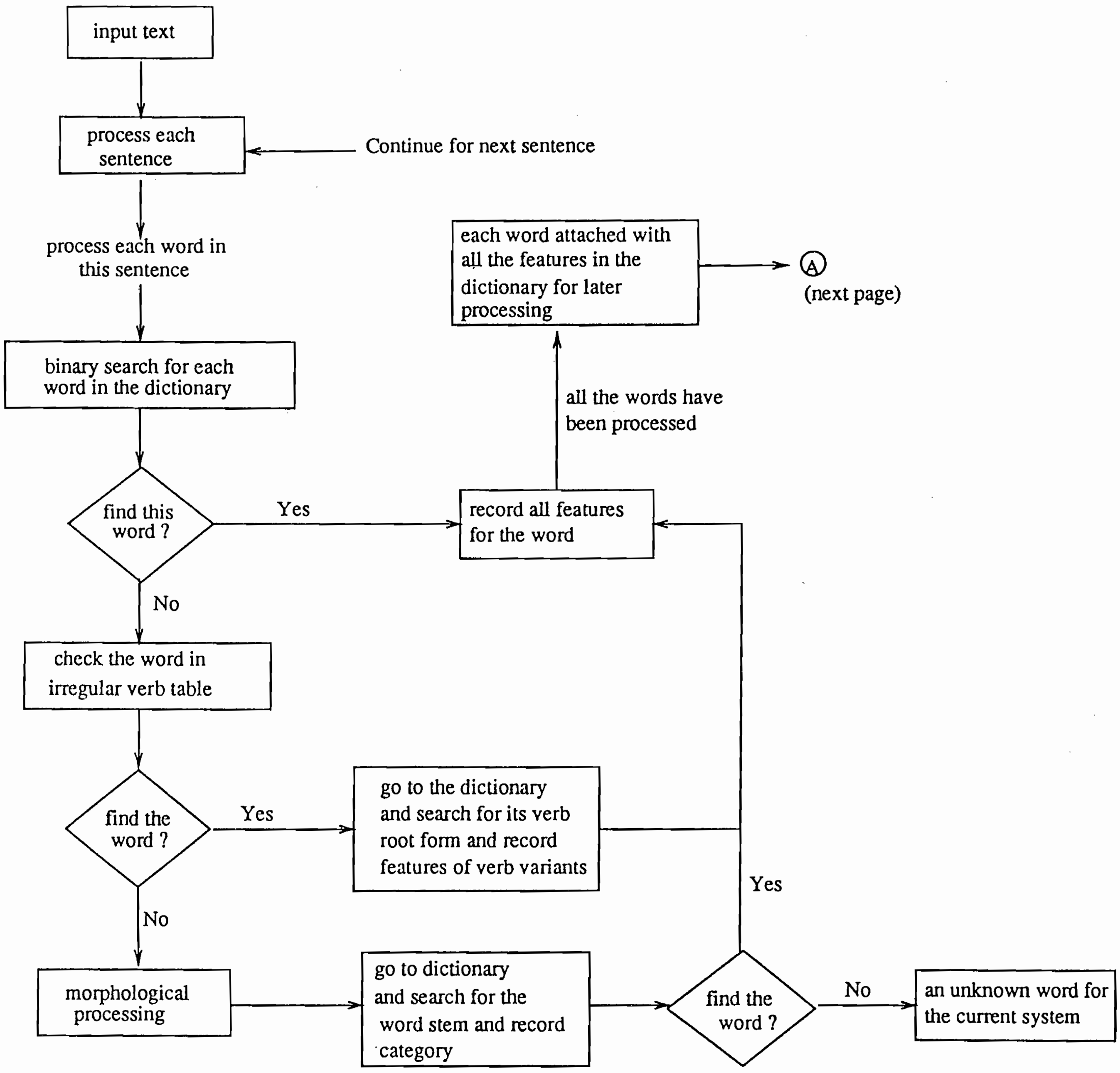
(VIII) using discourse strategies: Some of the errors concerning the scope of discourse such as anaphora may be too complex to be resolved in this project; however, we will explore the possible directions for future study.

Pattern matching, as an efficient technique from the programming perspective, has been shown limited in developing grammar checkers [see 10, for example]. Thus, a syntactic parser is one of the ultimate solutions to natural language understanding/analysis. To structurally analyze the input text, a top-down parser has been constructed. It was formulated in the augmented transition network (ATN) grammar [11]. To increase its precision of analysis, a set of word category disambiguation (WCD) rules has been devised to pre-process multiple word categories of some input words. The rules cut down the possibility of multiple word categories, and reduce the number of ambiguous sentence structures as well as processing time. For example, if a word has two categories, verb and adjective, and it is preceded by a determiner and followed by a noun, then the category, adjective is chosen such as falling in the falling rock. For the parser to be able to debug grammatical errors (besides judging whether the sentence is grammatical or not), two types of information have been encoded in the program: an expert model and a bug model. The expert model represents all the structural possibilities of correct sentences, whereas the bug model represents the error patterns we have formulated. For the expert model, a small segment of phrase structure rules by which we need to generate the structure of a correct sentence look like the following.

S -> NP VP
NP -> (Det) (AP) N ({PP, S'})
AP -> (Det) ("more") A {PP, S'}
VP -> V (NP) ({NP, PP})
PP -> P NP
S' -> Comp S
(S: sentence; NP: noun phrase; VP: verb phrase; Det: determiner; AP: adjective phrase; N: noun; S': embedded sentence; A: adjective; V: verb; PP: prepositional phrase; P: preposition; Comp: complementizer; ( ): optional symbol; {}: selectional symbol)

The bug model currently has three groups of error patterns: those manifested at noun phrase, verb phrase, and clause levels. Each of the groups is activated while the parser is analyzing/reconstructing its corresponding constituent. There are cases whose bug structure is unlikely to be represented, due, for instance, to its sporadic or idiosyncratic nature. In those cases, we map the expert model onto the input sentence and try to diagnose the nature of the problem by some devised heuristic. The dual-model mechanism is similar to a meta-rule concept described in Weischedel and Sondheimer [12].

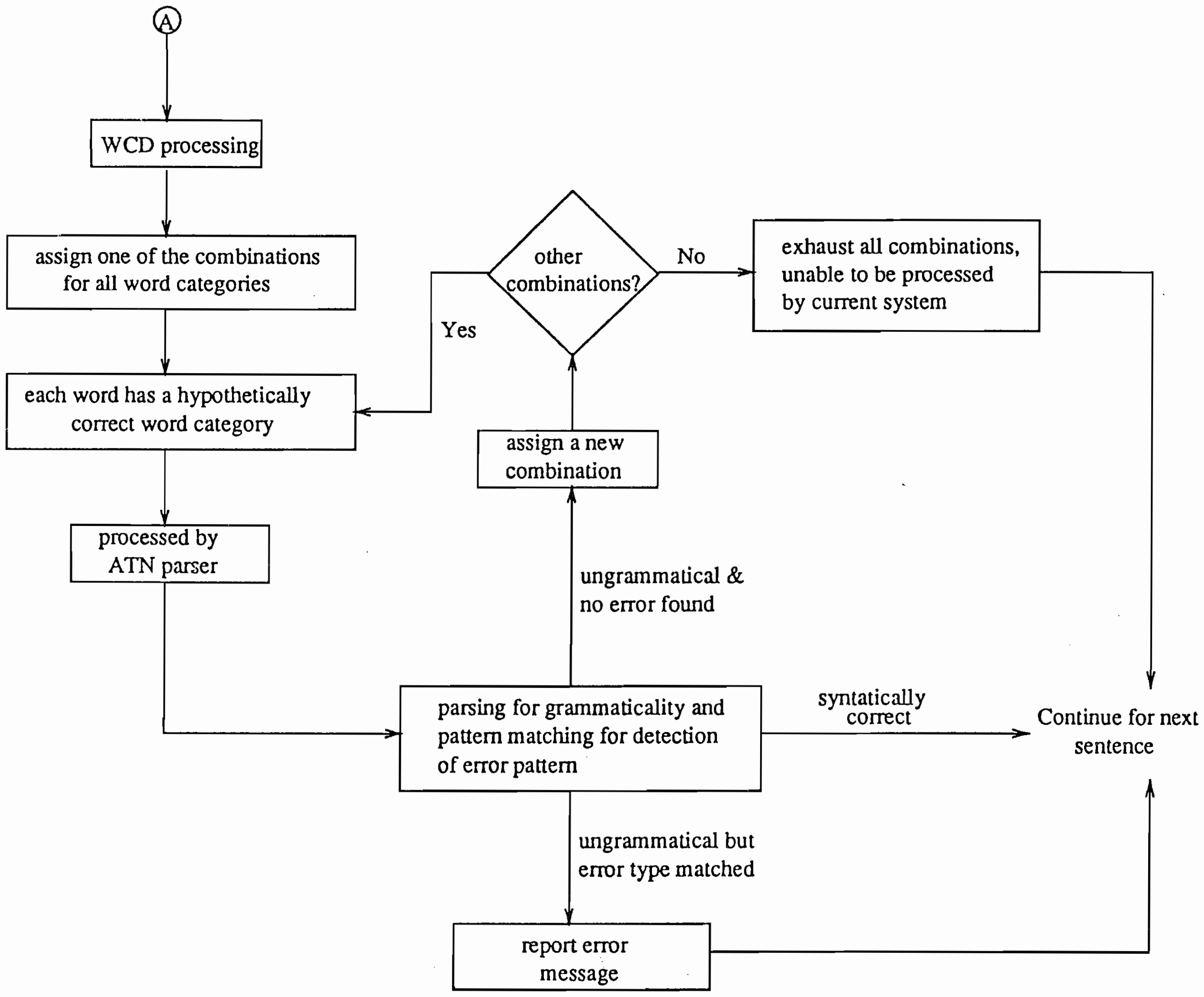
Figure 1 is a flow chart that demonstrates the procedures by which the grammar debugger processes each sentence and detects errors. The program allows regular English texts as its input and processes sentence by sentence. For each sentence, the program first uses the binary search algorithm to locate each word in the dictionary. If the program finds the word, it then records all associated features of this word. If the program fails, it proceeds to search for the word in the irregular verb table. If it finds the word, then the program goes to the dictionary to locate its root form and obtain features as well. If the program still can not find the word, it activates the suffix processor to do morphological processing. Notice that the category of a word before morphological processing is unknown and the word does not exist in the dictionary. After the word is processed by the suffix processor, it may be reformed and obtain its category information from this process. If the program still fails at this stage, the word is recognized as an unknown one for our current system. Up to this stage, each word, except unknown ones, is assigned its word category/categories and associated features. At the error detection level, i. e. after each word has been assigned categories, the program activates word category disambiguation (WCD) rules to cut down unlikely categories if a word has more than one category. After WCD processing, each sentence obtains a hypothetically correct combination of word categories to be processed by the parser. If the parser determines the sentence as grammatical, the program proceeds to the next sentence. If the sentence is determined as ungrammatical and detected by any of the error patterns, the program reports the error/feedback message and continues for the next sentence. If neither the parser nor pattern matching can determine the status of the input sentence, the sentence is assigned by another combination, if any, of word categories and the program repeats the parsing/pattern-matching processing. After the program exhausts all the possible combinations of word categories but still can not determine the status of the sentence (grammatical or ungrammatical) nor the nature of errors made, then the sentence is determined unable to be understood by the debugger/the current system.


input text
process each sentence
Continue for next sentence
process each word in this sentence
binary search for each word in the dictionary
find this word ?
Yes
No
record all features for the word
all the words have been processed
each word attached with all the features in the dictionary for later processing
A
(next page)
check the word in irregular verb table
find the word ?
Yes
go to the dictionary and search for its verb root form and record features of verb variants
No
morphological processing
go to dictionary and search for the word stem and record category
find the word ?
Yes
No
an unknown word for the current system

(to be continued)

Figure 1. The flowchart of processing a sentence in the program.

(to continue)


A
WCD processing
assign one of the combinations for all word categories
each word has a hypothetically correct word category
processed by ATN parser
other combinations?
Yes
No
exhaust all combinations, unable to be processed by current system
assign a new combination
ungrammatical & no error found
parsing for grammaticality and pattern matching for detection of error pattern
syntatically correct
Continue for next sentence
ungrammatical but error type matched
report error message

The operation of the grammar debugger is basically an interaction between the parsing and the matching of error pattern processes. Each sentence is presumed to be ungrammatical, or erroneous. The program thus activates the pattern matching process first. As previously mentioned, the bug model has three groups of error patterns. Whenever a corresponding constituent in a sentence is built by the parser, that group of error patterns is tested to match whether the input sentence has any of the error patterns. For example, the debugging process of sentence (11) can be illustrated in the following trace, run by our current system.

(11) No matter _ he say_, he like_ these job_.

Table 4

An Output Trace

```
=====================================================
Parse sentence : No matter he say, he like these job.

Searching in the dictionary ....
WORD     : no
CATEGORY : <av>
WORD     : matter
CATEGORY : <n v>
WORD     : he
CATEGORY : <ppn>
WORD     : say
CATEGORY : <v>
WORD     : he
CATEGORY : <ppn>
WORD     : like
CATEGORY : <v pp>
WORD     : these
CATEGORY : <d pn>
WORD     : job
CATEGORY : <n>

Using WCD-rules ....
WORD     :   no
CATEGORY :   <av>
```

```
WORD     :  matter
CATEGORY :   <n v>
WORD     :  he
CATEGORY :   <ppn>
WORD     :  say
CATEGORY :   <v>
WORD     :  he
CATEGORY :   <ppn>
WORD     :  like
CATEGORY :   <v pp>
WORD     :  these
CATEGORY :   <d>
WORD     :  job
CATEGORY :   <n>

Assigning category ....
no <av> matter <n> he <ppn> say <v> he <ppn> like <v> these <d> job <n>

Syntax Error !!   --->    No matter ......
no matter ( ? ) he say, he like these job.

Syntax Error !!   --->    Number disagreement: determiner -- noun
no matter he say, he like ( these ) ( job ).

Syntax Error !!   --->    Subject-verb disagreement
no matter ( he ) ( say ), he like these job.
no matter he say, ( he ) ( like ) these job.

This is not a correct sentence.  There are four errors.
==================================================
```

First of all, pattern matching of clause level errors is activated. Error types such as <u>although</u> ... <u>but</u> or <u>no matter</u> are classified under the clause level errors. This sentence matches the error pattern of <u>no matter</u>, which is thus flagged. Since there is only one noun phrase (NP), <u>these job</u>, error types at the noun phrase level are attempted and found matched with the type <u>determiner-noun disagreement</u>. The correct noun phrase should be <u>these jobs</u>, so <u>these job</u> is flagged. Subject-verb (S-V) agreement is checked for each NP and VP (verb phrase) in each clause. The program first locates the head of each NP and

VP and returns the number values (singular or plural) of both. Then, a comparison process is made to see whether they agree. In the case above, two incidents of S-V disagreement are found. If none of the error types are matched in any of the constituents, the parser proceeds and determines whether this is grammatical under our current phrase structure representation.

Currently, our checker can locate the following seven types of errors:

(1) <u>although</u> ... <u>but</u> combination

(11) <u>Although</u> he is poor, <u>but</u> he is happy.

(2) erroneous usage of <u>no matter</u>

(12) People can produce many things, <u>no matter</u> bad or good.

(3) determiner-noun disagreement

(13) We can know many <u>informations</u>.

(14) This is a book<u>s</u>.

(15) I like <u>an</u> book[2].

(4) unbalanced coordinated phrases

(16) He likes a dog but hate_ a cat.

(5) capitalization misuse

(17) There are not the exist of <u>T</u>elevision, computer, airplane, and so on.

(6) erroneous morphological changes in verb phrase, and

(18) I should <u>went</u> with you.

(7) subject-verb disagreement

(19) Human create_ the science.

(20) Human already <u>have</u> the ability to research the phenomena of space.

(21) But the development in science <u>have</u> bring great change.

(22) A man who like_ art like_ books.

---

[2] The initial phoneme of <u>book</u> is encoded in the dictionary.

Phase III. Phase three concerns what and how feedback messages should be given. When the program detects a grammatical error, giving appropriate feedback messages is essential for a grammar checker to achieve its educational goal. For this, we plan to design a message generating routine which basically matches a flag that is attached to each processing rule with a message file, and outputs the message to the users, possibly with some examples. The way we design the message is to use a template; namely, the message consists of some variables (as the underlined words in the following brackets) and literal texts (those in plain texts). For example, a feedback message for sentence (23) may look like that in the brackets.

(23) *The development in scientific technologies have bring great change.*

[development is the subject of the verb have. The subject is in 3rd person singular form. The following are 2 correct examples:

The baby in the living room watches television.

The lady who sits next to me teaches English.]

For technical terms, we consider using Chinese. In addition, the correction and feedback given should be set up with a user-friendly interface environment so that language teachers and learners will not encounter confusion -- which may seem reasonable or common to computer-literate people, though.

Future Research

As an exploratory but ambitious research study, the current project has its drawbacks to be improved. Since we are aiming to treat the errors manifested in natural English texts, the coverage of English grammar, of both correct and incorrect ones, is much wider than much of the previous research work. Thus, the error detection tasks are accomplished in an dissatisfying piecemeal manner. In the future, we, therefore, will try to formulate the global mechanism of the grammar debugger in a more generalized, from the linguistic perspective, framework. Possible directions we will refer to are those in Jensen, Heidorn, Miller, and Ravin [13], Kwasny and Sondheimer [14], Weischedel and Black [15], and

Weischedel and Sondheimer [see 12].

For the short-term goal, first, we will complete the analysis of the remaining corpus. Second, we will polish the programming tasks in detecting errors, as pattern matching is likely to fail for most of the error types and parsing is overloaded with problems in the long tradition of natural language processing. In addition, the grammar debugger's performance is still waiting to be tested. Last, we will consider at which point to give appropriate feedback messages and what actions allowed for the user to edit the mistake after the debugger detects an error.

References

[1] G. D. Price, *Grammatik IV: User's Guide,* Reference Software International, San Francisco, CA, 1989.

[2] S. Chen, L. Xu, "Grammar-Debugger: A Parser for Chinese EFL Learners," *CALICO Journal,* vol. 8, no. 2, pp. 63-75, 1990.

[3] G. Gazdar, E. Klein, G. Pullum, I. Sag, *Generalized Phrase Structure Grammar,* Harvard University Press, Cambridge, MA, 1985.

[4] B. Boguraev, T. Briscoe, "Large Lexicons for Natural Language Processing: Utilising the Grammar Coding System of LDOCE," *Computational Linguistics,* vol. 13, no. 3-4, pp. 203-218, 1987.

[5] B. Boguraev, T. Briscoe, Eds., *Computational Lexicography for Natural Language Processing,* Longman, London, 1989.

[6] R. J. Byrd, N. Calzolari, M. S. Chodorow, J. L. Klavans, M. S. Neff, O. A. Rizk, "Tools and Methods for Computational Lexicography," *Computational Linguistics,* vol. 13, no. 3-4, pp. 219-240, 1987.

[7] G. E. Heidorn, K. Jensen, L. A. Miller, R. J. Byrd, M. S. Chodorow, "The EPISTLE Text-Critiquing System," *IBM Systems Journal,* vol. 21, no. 3, pp. 305-326, 1982.

[8] A. Beale, "Towards a Distributional Lexicon," in *The Computational Analysis of English,* R. Garside, G. Leech, and G. Sampson, Eds., Longman, London, 1987, pp. 149-162.

[9] O. Stock, "Parsing with Flexibility, Dynamic Strategies, and Idioms in Mind," *Computational Linguistics,* vol. 15, no. 1, pp. 1-18, 1989.

[10] G. Hull, C. Ball, J. L. Fox, L. Levin, D. McCutchen, "Computer Detection of Errors in Natural Language Texts: Some Research on Pattern Matching," *Computers and the Humanities,* vol. 21, no. 2, pp. 103-118, 1987.

[11] W. A. Woods, "Transition Network Grammars for Natural Language Analysis," *Communications of the ACM,* vol. 13, no. 10, pp. 591-601, 1970.

[12] R. M. Weischedel, N. K. Sondheimer, "Meta-Rule as a Basis for Processing Ill-Formed Input," *American Journal of Computational Linguistics,* vol. 6, no. 3-4, pp. 161-177, 1983.

[13] K. Jensen, G. E. Heidorn, L. A. Miller, Y. Ravin, "Parsing Fitting and Prose Fixing: Getting a Hold on Ill-Formedness," *American Journal of Computational Linguistics,* vol. 9, no. 3-4, pp. 147-160, 1983.

[14] S. C. Kwasny, N. K. Sondheimer, "Relaxation Techniques for Parsing Grammatically Ill-Formed Input in Natural Language Understanding Systems," *American Journal of Computational Linguistics,* vol. 7, no. 2, pp. 99-108, 1981.

[15] R. M. Weischedel, J. E. Black, "Responding Intelligently to Unparsable Inputs," *American Journal of Computational Linguistics,* vol. 6, no. 2, pp. 97-109, 1989.

APPENDIX A

Descending Distribution of Errors in Main Types and Subtypes

MAIN TYPE	N	PER CENT
Det	326	19.65 %
Verb	231	13.92 %
Noun	178	10.73 %
PS	174	10.49 %
Concord	168	10.13 %
Sent	158	9.52 %
Prep	123	7.41 %
Lex	115	6.93 %
Conj	67	4.04 %
Mech	54	3.25 %
Adv	27	1.63 %
Adj	23	1.39 %
Pron	9	0.54 %
Aux	6	0.36 %
Total	1659	100.00 %

MAIN TYPE	SUBTYPE	N	PER CENT
Det	A-3	154	9.28 %
Noun	CN	129	7.78 %
Det	A-1	105	6.33 %
Lex	Dict	94	5.67 %
Prep	Prep-1	81	4.88 %
Concord	3S-1	75	4.52 %
Sent	Run-on	65	3.92 %
PS	PS-nadj	65	3.92 %
Verb	V-sub	59	3.56 %
Sent	Frag	57	3.44 %
Verb	VT-1	55	3.32 %
Conj	Conj-1	55	3.32 %
Verb	VT-3	51	3.07 %
Mech	Cap	49	2.95 %
Det	Det-a	49	2.95 %
PS	PS-adjn	41	2.47 %
Verb	VF	39	2.35 %
Concord	SV	39	2.35 %
Noun	UN	27	1.63 %
Adj	Comp-1	23	1.39 %
Prep	Prep-2	23	1.39 %
Concord	3S-4	21	1.27 %
Noun	NN	20	1.21 %
Prep	Prep-3	19	1.15 %
Concord	3S-5	15	0.90 %
PS	PS-nv	14	0.84 %
PS	PS-adjadv	12	0.72 %
Verb	V1	12	0.72 %
PS	PS-advadj	12	0.72 %
Det	A-2	9	0.54 %
Sent	E	9	0.54 %
PS	PS-vn	8	0.48 %
Concord	3S/paral	8	0.48 %

(to be continued)

(to continue)

Adv	ED	8	0.48 %
Sent	2S	8	0.48 %
Sent	Paral	8	0.48 %
Conj	NM	7	0.42 %
Verb	VT-2	7	0.42 %

MAIN TYPE	**SUBTYPE**	**N**	**PER CENT**
Pron	Pron-1	7	0.42 %
Adv	Adv-2	6	0.36 %
Concord	SP	5	0.30 %
Conj	AB	5	0.30 %
PS	PS-vadj	5	0.30 %
Adv	OS	5	0.30 %
Sent	Rel-1	5	0.30 %
Verb	V2	4	0.24 %
Aux	Aux-to	4	0.24 %
PS	PS-prepv	4	0.24 %
Det	Det-O	4	0.24 %
Lex	Dict-v	4	0.24 %
Lex	2V-1	4	0.24 %
Det	A-4	3	0.18 %
Adv	ASP	3	0.18 %
Mech	Ap	3	0.18 %
Lex	Dict-p	2	0.12 %
Verb	VT-4	2	0.12 %
Lex	Red	2	0.12 %
Sent	WH	2	0.12 %
PS	PS-adjv	2	0.12 %
Concord	3S-2	2	0.12 %
PS	PS-advconj	2	0.12 %
Adv	very/much	2	0.12 %
Verb	VT	2	0.12 %
Sent	Rel-3	2	0.12 %
Noun	One-N	2	0.12 %
Pron	anaf	2	0.12 %
Lex	SM	2	0.12 %
Concord	3S-3	2	0.12 %
Mech	Punct	2	0.12 %
PS	PS-conjprep	2	0.12 %
PS	PS-nadv	2	0.12 %
Lex	Sem-1	1	0.06 %
PS	PS-prepconj	1	0.06 %
PS	PS-N.PP	1	0.06 %
Lex	to/too	1	0.06 %
Lex	Dict-Es	1	0.06 %
Lex	A/E	1	0.06 %
Det	some/any	1	0.06 %
Det	Num-a	1	0.06 %
Concord	WS	1	0.06 %
Sent	Rel-2	1	0.06 %
PS	PS-infprep	1	0.06 %
PS	Red-Comp	1	0.06 %
Lex	PH	1	0.06 %
Sent	WHi	1	0.06 %
PS	N-adj	1	0.06 %

(to be continued)

(to continue)

Aux	Aux-2	1	0.06 %
Aux	Aux-1	1	0.06 %
Lex	Dict-mb	1	0.06 %
Lex	Dict-e	1	0.06 %
Adv	TA	1	0.06 %
Adv	SA	1	0.06 %
Adv	Adv-1	1	0.06 %
Total		1659	100.00 %

Appendix B

Verb Subcategorization

1	vp --> v	die
2	vp --> v np	love
3	vp --> v np pp[to]	give
4	vp --> v np pp[for]	buy
5	vp --> v np np	spare
6	vp --> v np pp[+loc]	put
7	vp --> v np s[fin]	persuade
8	vp --> v (pp[to]) s[fin] e.g. ... concede to the scientists that John has contact with the patient	concede
9	vp --> v s[bse] e.g. ... insisted (that) the job be given to John	insist
10	vp --> v (pp[of]) s[bse] e.g. ... require of them that they write a paper	require
11	vp --> v vp[inf] e.g. ... continue to be unhappy	tend
12	vp --> v vp[inf, +norm] e.g. I tried to leave.	try
13	vp --> v (pp[to]) vp[inf] e.g. ... seems (to us) to be unhappy	seem
14	vp --> v np vp[inf] e.g. ... believe John to be unhappy	believe
15	vp --> v np vp[inf +norm] e.g. ... persuade them to give themselves up	persuade
16	vp --> v (np) vp[inf +norm] e.g. ... promise Mary to do the homework	promise
17	vp[agr s] --> v np e.g. It bothered Li that Tom was chosen.	bother
18	vp[+it] --> v (pp[to]) s[fin] e.g. It seems (to us) that Mary is unhappy	seem
19	vp[agr np[there, PLU]] --> v np[PLU] e.g. There was a lion in the zoo. There were three wolves in the zoo.	be
20	vp --> v s[fin] e.g. Mary believes that it is true.	believe
21	vp --> v s[+Q] e.g. He inquired which way to go	inquire
22	vp --> v np s[+Q] e.g. Tell us why you did it.	tell
23	vp --> v pp[of]	approve

24	vp[+AUX] --> v vp[-AUX bse] e.g. I do like it be true.	do
25	vp --> v pp[to] pp[about]	talk
26	vp --> v adj/n e.g. I felt stupid/ a fool.	feel
27	vp --> v np adj e.g. They believe her guilty.	believe
28	vp --> v np np e.g. They consider this offer a big improvement.	consider
29	vp --> v v+ing e.g. She's given up smoking.	give up
30	vp --> v np v+ing e.g. They heard someone laughing.	hear
31	vp --> v np v-ed e.g. I want this work finished by tomorrow.	want
32	vp --> v np vp[bse] e.g. ... make her cry	make
33	vp --> v np pp e.g. ... compare it with a book.	compare

<u>Abbreviation</u>

fin: finite concede to the scientists that Jone has rings

bse: bare infinitive
insisted (that) the job be given to John

inf: infinitive continue to be unhappy

+norm means the noun is not in <u>it</u> or <u>there</u> dummy form

Q: question marker Tell us why you did it

AUX: auxiliary

agr: agreement

plu: plural