Robust Error Diagnosis and Self-Correction in Computer-Assisted Language Learning

Jiansheng Jiang, Tom Richards
Department of Computer Science and Computer Engineering
La Trobe University, Bundoora 3083, Victoria, Australia

Abstract: Diagnosing the errors in an input and providing effective remedy aid to assist the student to correct the errors are two important tasks of a useful CALL system. We propose a new approach — phrase structure expansion for language diagnosis in CALL, and suggest a new kind of knowledge representation formalism — Extended Place/Transition Tree which models a sentence structure in a way that particularly facilitates robust and automated sentence diagnosis and student self-discovery correction. The student's errors are detected through system-student interaction with an observation and a small amount of correct knowledge. An architecture based on these ideas is outlined and the performance of a prototype is briefly presented.

Introduction

A useful computer-assisted language learning (CALL) system must be able to diagnose the errors in a student's input and assist the student to correct the errors. Two important questions related to the development of such a system are:

- How can error diagnosis be performed truly robustly? It is desirable that the system is capable of detecting whatever errors encountered, rather than being limited to certain anticipated errors;
- How can error diagnosis be performed in such a way that its result will be useful for the student's remedial work?

Previous work in language error diagnosis relies either on the idea of "error anticipation" (e.g. Weischedel et al., 1978; Barchan et al., 1986), or on some *ad-hoc* methods, a typical case of which is Barchan and Wusteman's "intelligent guessing" (1988). Language analysis based on error anticipation can hardly be regarded as robust. As for "intelligent guessing", the designers admitted that "there is a point beyond which a set of words becomes too garbled to make a reasonable guess as to the intended structure" (5). This is not surprising since natural language is the most intractable, and as yet only partly understood phenomenon (Leech, 1987, 3).

Research on computer-assisted instruction (CAI) indicates that one good remedy assistance strategy is to offer context-related hints to guide the student to discover those errors her/himself (see below). Weischedel and Black (1980) claim that a system's capability to explain a parsing

THIS PAGE INTENTIONALLY BLANK

failure largely depends on the style and the structure of the parser. Therefore, in order to support the above strategy, the diagnostic method has to be carefully designed so that useful information can be collected during the diagnosis to assist the student's remedial work. Very little research that seriously takes CAI and learning theories into account has been done along these lines so far.

We shall describe here a research project designed to investigate the above two issues. Realising the limitations of previous approaches and based on careful study of existing methods for robust language analysis, we present a new system architecture — phrase structure expansion (PSE) — for language diagnosis in CALL, and suggest a new kind of knowledge representation formalism — Extended Place/Transition Tree (EPTT), which models a sentence structure in a way that particularly facilitates robust and automated sentence diagnosis and student self-discovery correction. This new approach enables the incorporation of a variety of AI techniques, resulting in an intelligent CALL architecture that achieves intelligibility, modularity, extendibility, and flexibility, and is capable of assisting the student in a more adequate manner.

We shall also describe a prototype system based on PSE architecture — Intelligent English Grammar Error Diagnoser (IEGRED). IEGRED differs from other CALL systems in several ways: firstly, it can provide highly robust analysis of both structural and syntactic ill-formedness without any error expectation; secondly, it can suggest reasonable actions to correct an ill-formed sentence; thirdly, it is very efficient. The evaluation of the system's performance shows the potential of PSE and EPTT in not only CALL but robust natural language processing (NLP) as well.

Error Diagnosis in Language Analysis

The analysis of ill-formed sentences is a topic studied in both the fields of CALL and robust NLP. The treatment of grammatical errors in a CALL system is different from the applications of NLP in other domains. The systems in those domains often do not bother to identify and explain grammatical errors if they do not cause misunderstanding problems (e.g. Granger, 1983). In a CALL system, it is the system's main task to pinpoint any language errors, generate clear error messages, and provide effective remedy aid. In general, error diagnosis in CALL is more difficult than in other contexts for several reasons: the first, all errors have to be detected correctly and reasons explained; the second, in most of the cases, little semantic and contextual information is available to assist the process of analysing ill-formed sentences (at least at the current stage); the third, effective remedy aid needs to be provided to assist the student to correct the errors. On the other hand, valuable lessons can be learnt from research in robust NLP. In this section, we shall discuss some techniques of dealing with ill-formedness attempted in both CALL and NLP, and present our own approach of robust language analysis in the context of CALL.

A useful CALL system has to consider three types of student errors: spelling errors, sentence structure errors, and syntactic concord errors. Spelling errors can be handled by using some readily available algorithms such as the one proposed by (Takahashi et al., 1990). The main task of a CALL system is thus to handle sentence structure errors and syntactic concord errors. Between them, it is evident that the former is the hard core: without clearly knowing what structural error(s) an input has, it is difficult for the system to decide which syntactic concord tests are applicable.

A common method used in CALL for language error diagnosis is "error anticipation". One of the typical systems that employ this technique is German Tutor (Weischedel et al., 1978), a language learning system for assisting students in developing their skills of reading comprehension and writing ability. In this system, a syntactic model of the language being taught is built by using the formalism of augmented transition network (ATN) (Woods, 1970). Frequently occurring errors in beginning German students are anticipated, and their incorrect

forms are added into the ATN grammar. The predicates associated with arcs in an ATN are designed such that they check whether the form is correct but will not block the transition. If the form is incorrect, an error message is recorded, and particular procedure associated with the predicate for generating hypothesis about the cause of the error is also called. Apparently, there are at least two problems with this method: the first, enhancing a normal parser with extra paths will make the logic of the grammar obscure, modification more dangerous, and debugging more difficult; the second, any unanticipated errors will cause the system to fail.

It has been observed in robust NLP that it is difficult for a parser working strictly from left to right to handle structural errors, since it can not take into account the right context of a possible structural error (e.g. the components appearing on *either* side of the problematic element). One such example is Weischedel and Sondheimer's "longest path" heuristics (1983). Given the sentence "He said that the snow the road" (from Mellish, 1989), the "longest path" heuristics can not indicate the simplest remedy action to make the sentence complete - adding a verb after "snow", but stops before "the road". This probably explains why most of the systems that can more or less robustly deal with structural errors organize the language analysis around some sort of patterns or frames. The examples include Kwasny and Sondheimer's ATN-based syntactic patterns (1981), Hayes and Mouradian's case-frames (1981), Granger's Schankian-like script-based situation descriptions (1983), Selfridge's conceptual-dependency-based frames (1986), and Mellish's generalized parsing rules and edges (1989). The potential and advantages of pattern matching in handling ill-formedness are well described by Hayes and Mouradian (1981): it enables a system to inspect the whole situation in which a problem element is located and select the best choice using some sort of goodness rating. There are two types of pattern matching: exact matching and tolerant matching. The latter is more useful in dealing with ill-formed utterances, since it allows for mismatch in certain parts of a pattern. The most flexible design in this respect is to allow for mismatch of any component in a pattern and have a facility for recording all mismatches that actually happened. The results can then be used for selecting a best-fit pattern among a group of competing alternatives by comparing the similarities between the input and each of the legal structures.

A declaratively expressed grammar such as context-free phrase structure grammar (CF-PSG) (Gazdar & Mellish, 1989) has the merits of generality, modularity, expressivity, transparency, and machine understandability. However, in such a grammar, the legal sentence structures are implicitly represented by separating them into individual rules, thus are not directly accessible by a pattern matching mechanism for sentence structure diagnosis. The hierarchical representation of a sentence structure by a set of rules even makes it difficult to modify a grammar to cope with structural errors. For example, if we try to insert an extra node into the following rules to enable them to deal with the sentence "Will the girl very like that hat?" ("very" is redundant),

S	--> aux NP VP	(1)
NP	--> det noun	(2)
VP	--> verb NP	(3)

should we add it in the rule (1) between *NP* and *VP*, or in (2) behind *noun* or in (3) before *verb*? Furthermore, no matter which way we go, we reduce the intelligibility of the grammar and introduce inefficiency to the parsing process. Following our earlier discussion on pattern matching for robust language analysis, it becomes clear that for handling structural errors, it is more effective to use those long somehow cumbersome sentence patterns (e.g. *auxiliary-verb det noun verb det noun* permitted by the rules above) than the original concise, modular phrase structures from which those patterns are generated, since the sentence patterns *explicitly* depict the relations between the constituents in a sentence (e.g. the *verb* should follow a *noun*).

To this end, from the perspective of handling ill-formed sentences, we need to expand the phrase structures in a CF-PSG into a set of explicit sentence patterns. With the set of patterns, an input sentence's intended structure can be recognized by comparing it with those patterns, and the sentence can then be analysed according to that structure. We call this approach *phrase structure expansion* (PSE).

The number of sentence patterns derivable from a grammar is usually large (George, 1972, 20), and some patterns can be complex. Hence, it is preferable that the patterns are created by the system automatically and they cover "typical phrases" that would be most likely input by the particular group of students who are going to use the system. Making sentence patterns user-oriented will also improve the efficiency of the system considerably (Samuelsson & Rayner, 1991).

Based on this understanding, we suggest a novel architecture — PSE architecture, which allows student-oriented sentence patterns to be generated by an explanation-based learning (EBL) facility from a set of training sentences most likely input by the intended users. Given an input sentence, a best-fit module will select a pattern that best captures its intended structure. A hierarchical model based on the Extended Place/Transition Tree (EPTT) formalism for analysing sentences with that particular structure is then generated by the system and used to diagnose the input sentence. The outline of the architecture and the details of EPTT will be given later.

Remedial Work in CALL

Errors may help a student to develop her/his problem-solving ability and increase her/his knowledge, but this can only happen when the student has enough information to determine what caused the error and correct it. In such a case, the errors are said to be *constructive*, otherwise, they are *non-constructive* (Burton & Brown, 1982). One of a language learning system's tasks is to provide the student with adequate information so that non-constructive errors can be transferred to constructive ones.

Cohen (1985) points out that the time at which a computer tutor needs to provide feedback depends on several factors, such as the mastery level of students, the short-versus-long term retention requirements of the learning, and the availability of prior knowledge to support learning. Cohen (1983) also suggests that feedback should state *whether* the answer is correct or not and explain *why* the answer is correct or incorrect. The research on feedback done by Kulhavy (1977) indicates that, in general, negative feedback — the feedback provided when the response is incorrect — is more helpful than positive feedback. Negative feedback informs the student that previous responses must be modified to prevent the response from recurring.

Work done on intelligent tutoring systems (ITSs) such as WEST (Burton & Brown, 1982) and LISP Tutor (Reiser et al., 1985) suggests that it is better to offer the student several levels of hints rather than actual solutions. WEST, a computer coach that provides help to a student playing an open-ended game, maintains a set of pedagogical principles which offer not only theoretical justifications for the techniques it uses, but also heuristic pedagogical guidance in dealing with the student. The following two principles explicitly state how a tutor should give help and what to do afterwards:

Principle 10: If the student asks for help, provide several levels of hints.
Principle 8: After giving advice to the student, offer him a chance to retake his turn, but do not force him to.

Similarly, LISP Tutor has been designed "to provide only as much guidance as necessary while encouraging the student to generate as much of the solution as possible. Thus, the tutor generally

tries to provide hints rather than actual solutions" (Reiser et al., 1985, 10). System guidance is offered at several levels. Hints towards correct solutions are also provided in the form of queries and reminders of the current goals.

It should be pointed out that although feedback given by a tutor should be able to determine whether an answer is wrong, and if it is, why, it is not necessary to say the two things at the same time. In fact, sometimes it is more helpful if the system can indicate the incorrectness of an answer but provide the student with the opportunity of discovering why it is wrong. In the ITS SOPHIE (Brown et al., 1982), for example, it has been realised that although the explicit explanation given by the expert when it locates a fault in a circuit can quickly meet the student's local request to understand the process for solving a particular problem, it is not good for helping the student to build a mental model about the global features of the domain.

Turning back to remedial work in language learning, Johnson argues that "where practice of known language is carried out by learners working alone, such practice should be self-correcting in that it should not need a teacher to correct every response the learner makes" (Johnson, 1973, 182). George (1972) points out that in order for a student to correct an incorrect form with a correct substitution (e.g. "he plays" for "he play"), s/he must be able to see the distinction between them. The teacher's task is therefore to direct the student's attention to the item (the word "he" in the above example) that will help her/him to see the *distinction* between the incorrect form and the wanted form.

Based on the above points about feedback and remedial work in language learning, we have designed a strategy for providing feedback and remedy guidance used by our system. This strategy is called "gradually refined context-error association for self-discovery correction". Using this strategy, when the student has made some errors in an answer, the system first informs the student about the existence of the errors, then encourages the student to discover errors her/himself by providing several levels of hints, but leaving the option of immediately inspecting those errors open. To direct the student's attention to the relevant context in which the errors occurred, the system displays those sentence parts that contain errors in a gradually refined manner. Explanations as to the grammar rules related to individual errors can also be inspected. The student is also given the option of exiting from the remedial process and rewriting the answer or proceeding to the next exercise.

The above remedial strategy makes extensive use of the diagnostic results from the language analysis process. As already mentioned, how much information is available from language analysis process for feedback purpose is affected by the style and structure of the grammar, as well as the techniques utilized. Therefore, the language analysis facility needs to be designed such that it will support our strategy. The solution lies on the design of EPTT.

PSE Architecture

The general architecture is given in Figure 1. This architecture consists of three main modules: an EBL module, a best-fit selection module, and a language analyser.

Explanation-based learning is a machine learning technique. Given correct domain knowledge, a system based on EBL can learn justified generalizations from examples. "Typically, the purpose of EBL is to produce a description of the concept that enables instances of the concept to be recognized efficiently" (Minton et al., 1989, 64). The use of EBL technique allows us to have the control over the types of patterns to be generated, and to generate those patterns in a systematic manner. The EBL module in our PSE architecture consists of a parser and a pattern derivation subsystem. The parser parses each of the training sentences, then passes the parse tree to the pattern derivation subsystem, which will derive a sentence pattern together with the

indexing numbers of the phrase structures used by the parser to parse the sentence in the grammar. The phrase structures derived are stored in a right order that facilitates their later connection to form a hierarchical model for analysing the sentences with the structure they concern. The training sentence, the sentence pattern derived and the set of phrase structure indexing numbers are collectively called a *sentence template*. If we view the whole data structure as an object in the sense of object-orientation, these data can be held in the slots of a sentence template object conveniently. Sentence templates are stored in the template bank and organized into groups indexed by both the lengths and the partial syntactic structures considering certain syntactic features (e.g. whether it is declarative or interrogative? affirmative or negative? simple present tense or past perfect continuous? etc.) of their patterns to facilitate quick access.

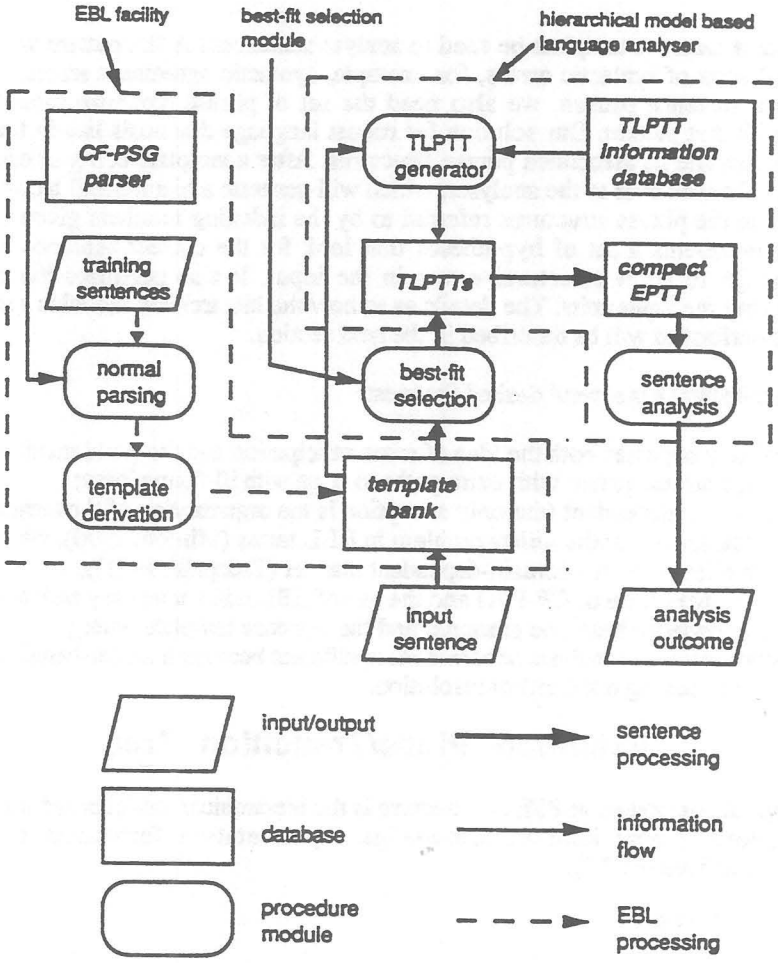

Figure 1

Ideally, if concurrent processors were available, patterns would be examined in parallel in the best-fit selection process, since there is no order among them and pattern selection would be very fast. Currently, however, patterns still have to be tested sequentially. Samuelsson and Rayner (1991) propose two methods of selecting patterns for well-formed sentences: the decision-indexing method and the key indexing method. Unfortunately, neither will work on ill-structured input. We

present a three-pass best-fit selection method. In the first pass, the best-fit selection module performs a partial pattern selection guided by the information provided by the input (its length plus permitted tolerance and its partial structure concerning certain syntactic features) and the index of the template bank to narrow down the search space. In the second pass, it then applies a full but tolerant pattern matching to the candidates from the first pass by computing the similarities between the candidate templates and the input sentence, using the longest common substring (LCS) formula (e.g. Hunt & Szymanski, 1977). If a unique template gets the highest score for the similarity, it is the best-fit one for the input. In case there is a tie, the system then turns to the student by presenting the training sentences associated with the candidate templates and asking for her/his choice. In CALL, such system-student cooperation has been demonstrated to be an effective way to bypass unsolved difficulties in NLP and expensive computation (Nyns, 1990).

How can a sentence template be used to analyse sentences? A flat pattern will not be able to deal with all sorts of syntactic errors, for example, syntactic agreement errors. This is why, in addition to a sentence pattern, we also need the set of phrase structure rules for processing sentences with that pattern. Our solution for robust language diagnosis lies in the use of both a sentence pattern and its associated phrase structures. After a template being selected for an input sentence, it is handed over to the analyser, which will generate a hierarchical model in the form of an EPTT from the phrase structures referred to by the indexing numbers given in the template. The model represents a set of hypotheses that look for the correct sentence structure but is flexible enough to allow structural errors in the input. It also performs the analysis of the relations among the input units. The details as to how the hierarchical model is generated and the diagnosis is performed will be described in the next section.

This architecture has several desired features:

- It completely bypasses both the idea of error anticipation and the problematic method of equipping a normal parser with extra paths to cope with ill-formedness;
- It is language independent (the only exception is the organization of the template bank to allow quick access, or the *utility* problem in EBL terms (Minton, 1990), which has to be dealt with effectively in a domain-dependent manner (Tadepalli, 1991));
- The declarative nature of CF-PSG and the use of EBL make it an easy task to extend the system's knowledge base (the grammar and the sentence template bank);
- In run-time, sentence analysis becomes more efficient because a model-based analyser requires neither backtracking nor conflict resolution.

Extended Place/Transition Tree

The central component in PSE architecture is the hierarchical model based language analyser, which employs a new kind of knowledge representation formalism called Extended Place/Transition Tree (EPTT).

Expanded EPTT

There are two forms of EPTT: *expanded* and *compact*. An expanded EPTT consists of a set of *three level place/transition trees* (TLPTTs), each of which represents a syntactic rule in a CF-PSG. A TLPTT is a directed graph with two types of nodes, namely *places* and *transitions*. The root of a TLPTT is a place representing the non-terminal on the left hand side of a rule. The bottom of a TLPTT is a set of places representing the categories on the right hand side of the rule. To represent a syntactic rule in a language having word order such as English, the order of the places at the bottom of a TLPTT must be retained. If a place at the bottom represents a non-terminal, it is called a *tree place*, otherwise, it is called a *primitive place*. A tree place itself

contains a TLPTT, since a non-terminal in a parse tree will have other categories as its own children. Between the root node and the bottom nodes of a TLPTT, there is a transition. Transitions can be divided into two types: *plain transition* with one and only one child place, and *K-cluster transition* having more than one place descending from it.

Consider the sentence "the boy eats that juicy orange." The structure pattern of this sentence is: *article noun verb demonstrative-adjective adjective noun punctuation*, which can be generated from the Gazdar-Mellish (1989) phrase structures shown below:

```
S-maj        --> S-decl final-punc                          (1)
    <S-decl first-word capitalization> = yes                (2)
    <final-punc type> = period                              (3)
S-decl       --> NP1 VP                                     (4)
    <NP1 head per> = <VP head per>                          (5)
    <NP1 head num> = <VP head num>                          (6)
NP1          --> art noun                                   (7)
    <art num> = <noun num>                                  (8)
    <art pho> = <noun pho>                                  (9)
VP           --> verb NP2                                   (10)
NP2          --> demon-adj [adv] [adj] noun                 (11)
    <demon-adj num> = <noun num>                            (12)
```

To increase the generality of these phrase structures, we have added the category *adv*; which gives an example of *optional* node in an EPTT. The square brackets [] means none or one or more. Therefore, an EPTT node can also accommodate a sequence of words with a *repeated* category. Meanwhile, the use of some categories in the rules is for simplicity: for example, subcategorization of some categories such as *verb* and *noun* are necessary in a real implementation.

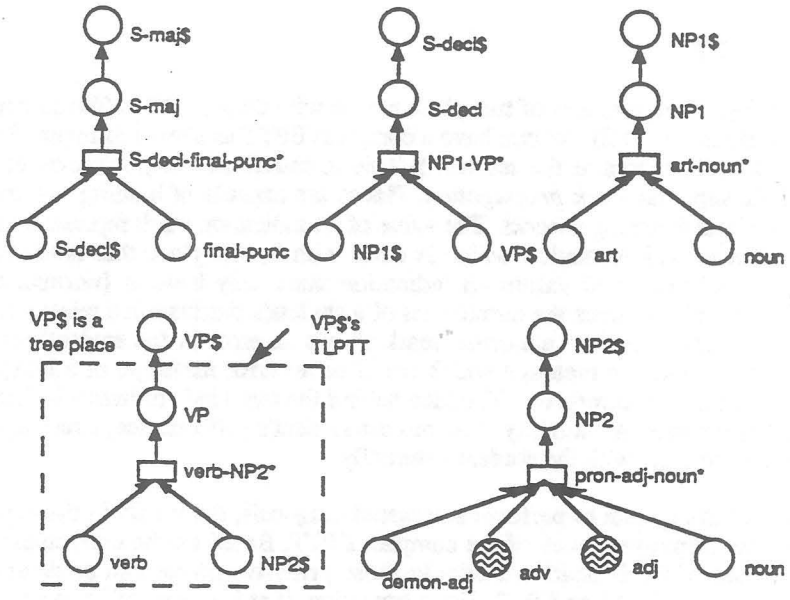

Figure 2

The tree places corresponding to the above phrase structures are shown in Figure 2. In that figure, places are represented as circles and transitions as rectangles. The places filled with curved lines are those whose contents are both omissible and repeatable. Symbols ending with "$" and "*" are tree place names and transition names, respectively. Other symbols are the names of either root places or primitive places. The set of features that associated with a particular phrase structure are converted into the rules in the local rule base stored in the corresponding transition or primitive place. For instance, the transition *NP1-VP** will have a local rule base consisting of two rules corresponding to the features (5) and (6). Those local rule bases are not given in the graphic representation of TLPTTs, but we need to be aware of their existence. Each rule in a local rule base is essentially an IF-THEN-ELSE statement. If the condition, which represents a piece of correct syntactic knowledge (e.g. the noun in the subject and the verb have the same number), is satisfied, the system will record that the student has made a correct attempt to apply this piece of knowledge; otherwise, actions will be taken to issue appropriate error messages or remedy hints, etc. Potentially, such information can be very useful for a system to decide further teaching contents and strategies.

If an input contains structural errors, some rules for examining syntactic constraints may not be applicable. The applicability of a rule in a particular situation is determined by the values, which are words or sentence parts retrieved from relevant places, of the variables the rule possesses. Therefore, a rule for examining the number concord between an article and a noun will not be applied if one of them is missing. Determining the applicability of a rule based on the state of the relevant sentence parts is an important part of robust language processing.

During execution, a set of phrase structures are transferred into a set of tree places containing TLPTTs by a system automatically, using the data from the TLPTT information database such as the appropriate rules for syntactic concord tests etc. The transfer involves some complex computation, such as the identification of operational goals (Samuelsson & Rayner, 1991), automatic symbol generation, and rule and procedure coupling, whose details are not discussed here.

Compact EPTT

By replacing all occurrences of tree place names with their TLPTTs (the structures under the tree place names in Figure 2), we can have a complete EPTT as shown in Figure 3 (disregard the stars and dots in the places at the moment). This is called the *compact* form of the EPTT. A compact EPTT supports *mark propagation*. Places are capable of holding two types of marks, *indication marks* and *message marks*. The value of an indication mark represents the state of the piece of student-input information which is dealt with by the place that holds that mark. The simplest design of the set of values an indication mark may have is {normal, abnormal}, in which a normal mark indicates the correctness of a student's performance related to one aspect of the domain knowledge, and an abnormal mark shows an error in the student's performance. A message mark is actually a message which could be an error message, or a hint for assisting a student to correct her/his errors, etc. The idea behind the two kinds of marks is that an indication mark is used by the system internally to record the student's performance; a message mark is used by the system to interact with the student externally.

In order for the system to perform automated diagnosis, the words in the input are inserted into the relevant primitive places of the compact EPTT. Based on the correctness of the words, the proper values of the indication marks in those primitive places can be determined. This is called the *initial marking* of an EPTT. For a transition, if at least one of its input places contains a mark, then a mark propagation action can be taken, which involves adding a mark with the same type into the output place of the transition and results in a *new marking* of the EPTT. A

normal mark will be added into the output place of a transition if and only if the following conditions hold:

- every mark with the same type in the input places of the transition whose contents are not omissible has the value *normal*;
- if the transition is a K-cluster transition, no error in the student's input has been identified by the local rule base associated with it.

As an example, Figure 3 shows an EPTT that is given the sentence "the boy that eat oranges", with both the initial marking and the final state of the mark propagation. A dot represents a normal indication mark and a star an abnormal indication mark. Abnormal marks in places can be caused by various reasons. For example, in Figure 3, *art* has an abnormal mark because "the" has not been capitalized; *S-decl* has an abnormal mark partly because its child places contain abnormal marks and partly because "boy" and "eat" have different numbers; *demon-adj* has an abnormal mark because the word for this place is missing, etc. This gives rise to the need of distinguishing different kinds of errors discussed below. Note that in order to accommodate the redundant word "that" between "boy" and "eat", an extra primitive place *epp1* has been created and inserted in the EPTT in the right position.

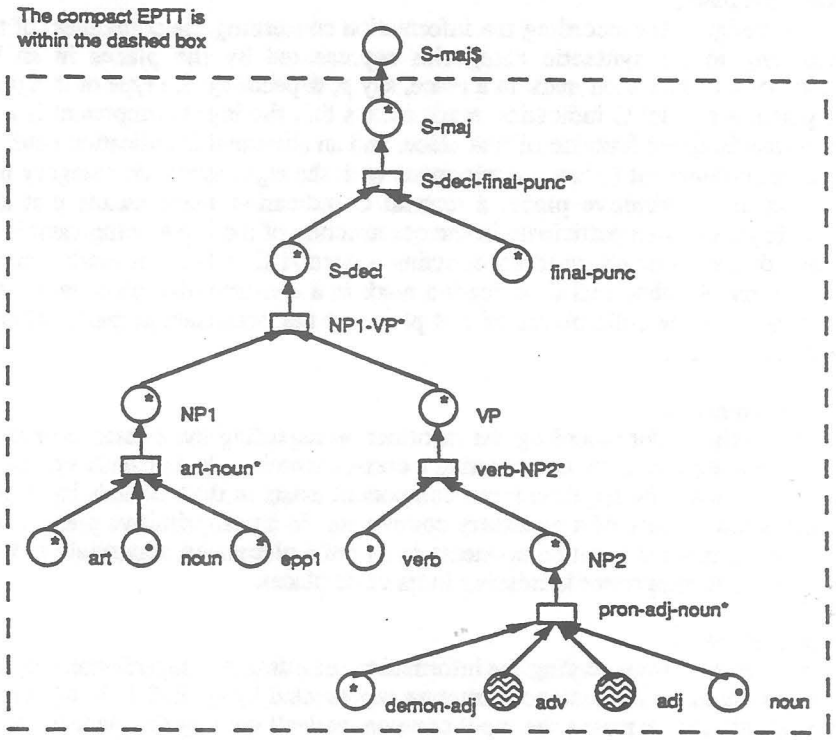

Figure 3

Clearly, the diagnostic process is entirely based on the encoded knowledge about the requirements of *correct* sentences. In such a system, any violations to a correct model, including the structure it represents and the constraints it encodes, are treated as errors.

Dealing With Different Kinds of Errors

Sentence structure errors and syntactic concord errors can be divided into several types:

1. an input component fits into the place but does not have the required feature(s); for example, a comma for the final punctuation of a complete sentence;
2. a required input component is missing; for example, in "in morning", a determiner is missing;
3. there exists a superfluous input component according to the selected EPTT; for example, the word "him" in "Mary married the man whom she wanted to marry him.";
4. a group of input components are incorrectly related together to construct a higher level input component, which can be further divided into two subcategories:

(1) there exists mismatch between two components, for example, the mismatch between the number of a noun in the subject of a sentence and that of a verb;
(2) these components are put together in a wrong order.

In order to deal with these types of errors, we define the following three kinds of indication marks:

1. C indication marks

They are designed for recording the information concerning the correctness of the student's input according to the syntactic categories represented by the places in an EPTT. The interpretation of a C indication mark in a place, say *p*, depends on the type of that place. If *p* is a primitive place, a normal C indication mark means that the input component is a correct one according to the designed function of that place, and an abnormal C indication mark indicates an incorrect input component (often a component with the right syntactic category but incorrect form). If *p* is not a primitive place, a normal C indication mark means that all the input components in *p*'s children participate in the construction of the input component in *p* correctly. This implies that each of *p*'s children contains a normal C indication mark and they are put together correctly. An abnormal C indication mark in a non-primitive place means the group of input components in the child places of that place are not combined properly (often containing syntactic concord errors).

2. M indication marks

They are designed for recording the information regarding the existence or absence of the necessary input components for constructing a correct sentence. In a primitive place, a normal M indication mark means the required input component exists in the student's input. An abnormal mark indicates the absence of a necessary component. In a non-primitive place, a normal mark means there is no missing input components in its child places. An abnormal mark means there is at least one input component is missing in its child places.

3. S indication marks

They are designed for recording the information regarding any superfluous components in the student's input based on the sentence structure represented by an EPTT. In a primitive place, a normal S indication mark means the input component dealt with by the place is not superfluous. An abnormal mark (it can only appear in an extra primitive place created and inserted into the corresponding EPTT by the system) indicates a superfluous component. In a non-primitive place, a normal mark means there is no superfluous input component in its child places. An abnormal mark means there is at least one such a component in its children.

To deal with these three types of indication marks, we name three types of vectors:

- C-vector, which represents a marking of C indication marks in an EPTT;

- M-vector, which represents a marking of M indication marks;
- S-vector, which represents a marking of S indication marks.

A C-vector is denoted as Vc. An M-vector is denoted as Vm. An S-vector is denoted as Vs.

In the sentence "the boy that eat oranges" given earlier, a superfluous word "that" before "eat" can be recognized by an EPTT, resulting in an extra primitive place *epp1* being inserted into it. The initial markings of the vectors are as follows. Again, a dot represents a normal indication mark and a star an abnormal indication mark. A place containing no mark is represented as 0.

Vci = ((0 S-maj) (0 S-decl) (0 NP1) (* art) (• noun) (0 epp1) (0 VP) (• verb) (0 NP2)
(0 demon-adj) (0 adv) (0 adj) (• noun) (0 final-punc))
Vmi = ((0 S-maj) (0 S-decl) (0 NP1) (• art) (• noun) (0 epp1) (0 VP) (• verb) (0 NP2)
(* demon-adj) (0 adv) (0 adj) (• noun) (* final-punc))
Vsi = ((0 S-maj) (0 S-decl) (0 NP1) (• art) (• noun) (* epp1) (0 VP) (• verb) (0 NP2)
(0 demon-adj) (0 adv) (0 adj) (• noun) (0 final-punc))

The place *demon-adj* expects a demonstrative adjective, but received none, resulting in an abnormal M mark in it, so does the place *final-punc*. The star in the sixth position in Vsi reflects the existence of the superfluous word.

If a superfluous word is in one of the input places of a transition, two different versions of the sentence part will need to be kept and transferred to the output place of the transition: one with the superfluous word, one without it. This allows the rules for testing syntactic concord to be applied to the right parts of the sentence. In this example, the place *NP1* will contain both "the boy" and "the boy that", but the rules for checking the agreement between an article and a noun will only be applied to "the boy".

The final markings of the three vectors are

Vcf = ((* S-maj) (* S-decl) (* NP1) (* art) (• noun) (0 epp1) (• VP) (• verb) (• NP2)
(0 demon-adj) (0 adv) (0 adj) (• noun) (0 final-punc))
Vmf = ((* S-maj) (* S-decl) (• NP1) (• art) (• noun) (0 epp1) (* VP) (• verb) (* NP2)
(* demon-adj) (0 adv) (0 adj) (• noun) (* final-punc))
Vsf = ((* S-maj) (* S-decl) (* NP1) (• art) (• noun) (* epp1) (• VP) (• verb) (• NP2)
(0 demon-adj) (0 adv) (0 adj) (• noun) (0 final-punc))

Since the first element in Vcf has a star, the sentence contains some syntactic concord error(s). A top-down trace will reveal those errors: a non-capitalised first word "the" and the mismatch between "boy" and "eat". Similarly, a system can also work out the missing component and superfluous component errors. The detection of both a missing word and a superfluous word and the recognition of the same category they have allow the system to make a further hypothesis that the sentence may have a pair of words out of order. It can then present the hypothesis to the student for confirmation or rejection.

System-Assisted Self-Discovery Correction

In the course of diagnosis, any errors detected will cause some error messages to be stored in the relevant place. In the case of a K-cluster transition, the sentence parts (words, phrases) involved are combined and transferred to the output place of the transition, forming a new constituent at a higher level for further analysis. For example, the place *VP* in Figure 3 will contain the sentence part "eat oranges". In this way, an error message is kept with the context in which the related error occurred, together with other information such as explanation and error

hints, etc. The resulting diagnostic information establishes a good basis for the system to provide "intelligent" guidance to the student for her/his own remedy actions in a context-sensitive manner. Based on the chain of the causal relations between the indication marks in a parent and that in its children, the system could provide a trace of an error from the most abstract level to the most specific level and present error messages with varying degrees of abstraction (e.g. from "There is an error in the sentence part 'eat that oranges'" to "There is a number mismatch error in the sentence part 'eat that oranges'" to "The number of the demonstrative adjective 'that' and the number of the noun 'oranges" are not the same"), which supports our remedial strategy of "gradually refined context-error association" for self-discovery correction we mentioned earlier.

Intelligent English Grammar Error Diagnoser

To evaluate our PSE architecture and EPTT formalism, we have developed a prototype CALL system — Intelligent English GRammar Error Diagnoser (IEGRED).

The Architecture and Application Modes

The architecture of IEGRED is shown in Figure 4, in which the components drawn as rectangles represent various databases, those drawn as round rectangles represent programs for processing data, and the components within rhombic boxes are the input to and the output of the system. The arrows show the directions of information flow. It can be easily seen that the architecture of IEGRED is based on PSE architecture presented earlier, with the addition of several other components for storing information required by an object-oriented system (e.g. the class definitions and the procedures for creating the objects in those classes), for dealing with input/output, and for implementing user interface. To increase the system's practical applicability, IEGRED also has two extra components whose details are not discussed here: Word Guesser for guessing the category of a word not in the dictionary, and Exercise Generator that generates drill exercises automatically for English grammar practice.

The system has a built-in dictionary with about 500 words organized into a class hierarchy containing 40 classes. Multiple senses of a word are allowed. For instance, the word "can" can be used either as an auxiliary verb (e.g. in "that can make him sick"), or a noun ("that can made him sick"). The CF-PSG it has contains about 150 phrase structures concerning various types of sentences with different tenses and voices. In its EBL module, it employs a chart parser (Kay, 1976; Gazdar & Mellish, 1989) to parse training sentences.

A TLPTT or a compact EPTT is represented as a list, which provides the flexibility for the system to modify an EPTT during run-time to accommodate redundant components in the student's input. Places and transitions are defined as instances of different classes, with slots for holding various kinds of data and procedures, such as a TLPTT, a local rule base, a mark, etc. Rules in rule bases are created and maintained independently from their calling places or transitions, and a rule can be used anywhere necessary.

IEGRED does not anticipate any kinds of error, and it uses no semantic knowledge in its language analysis process. However, IEGRED is able to perform sophisticated diagnostic activities and to provide instructive feedback (refer to the Appendix). IEGRED has two application modes: *diagnosis mode* and *drill mode*. The fundamental part of the system is, of course, the diagnosis facility, which is capable of analyzing sentences input freely by a student. However, together with other modules such as Exercise Generator, this facility is exploited to support a drill lesson for practising grammar knowledge. When in the drill mode, a student can decide her/his own drill pace (fast, normal, or slow). The designer or the teacher can include or exclude an exercise when a certain pace is considered by simply specifying the value for the pace slot of the corresponding exercise object. Similar to the diagnosis mode, if a student has made a

mistake, s/he can try to find it out under the assistance of the system. Then the student can either re-enter the answer or go to the next exercise.

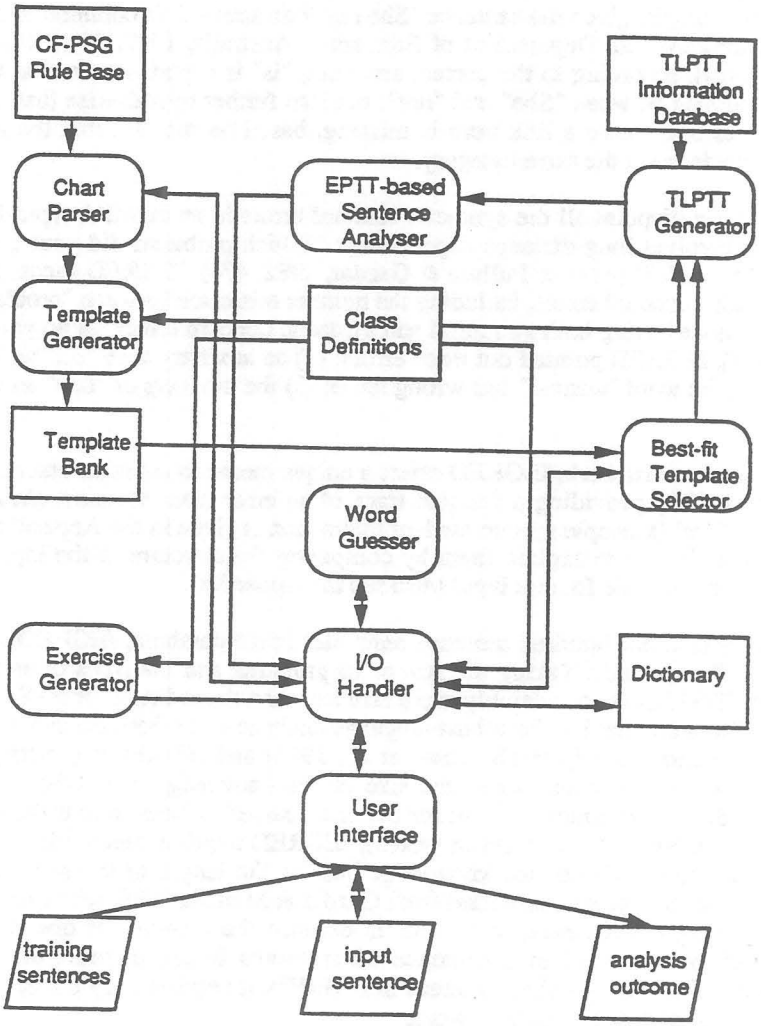

Figure 4

The Evaluation

We have evaluated IEGRED's performance in four aspects: its capability to recognize the intended structure of a sentence involving structural errors, its capability to detect various kinds of syntactic concord errors, its capability to provide effective remedial aid, and its efficiency.

Experimental results have shown that in all of the four aspects, IEGRED performs quite well. Given a sentence with multiple and arbitrary structural errors, IEGRED can always suggest a sentence structure that often indicates minimum changes to make the sentence correct. For example, given the sentence "Mary girl.", IEGRED pointed out that two words, a verb and a determiner, were missing. Similarly, IEGRED suggested that "Take I letter see them." can be

corrected as "I" + "take" + a determiner + "letter" + the preposition "to" + "see" + "them" + "." ("+" here means "followed by"). IEGRED can also find out a wrong word order error (which is an extremely difficult task to most of the existing CALL systems based purely on syntactic analysis). For example, given the sentence "She *not is* an actress." (a common error among Greek students (Commonwealth Department of Education, Australia, 1982, 29)), IEGRED could not only find out that, according to the correct structure, "is" is superfluous and at the same time a link verb is missing between "She" and "not", but also further hypothesize that "is" should have been in the position where a link verb is missing, based on the fact that the missing and the superfluous words are of the same category.

IEGRED can pinpoint all the syntactic concord errors in an input. A typical example is the sentence that involves long-distance dependency: "Which problems did your professor say she think was unsolvable?" (refer to Pullum & Gazdar, 1982, 474). IEGRED successfully diagnosed all three syntactic concord errors, including the number mismatch between "problems" and "was". Given the sentence having both structural and syntactic concord errors "Who you wanted to give this books to?", IEGRED pointed out three errors: (1) an auxiliary verb "do" with a correct form is missing; (2) the word "wanted" has wrong tense; (3) the numbers of "this" and "books" do not match.

Regarding remedial work, IEGRED offers a unique means to assist the student to discover the errors her/himself by providing a detailed trace of an error from the most abstract level to the most concrete level (a sample system-student interaction is given in the Appendix). For structural errors, IEGRED is able to explain them by comparing the structure of the input with a correct sentence structure suitable for that input (also see the Appendix).

With more than one hundred sentence templates in the database, IEGRED usually responds within one to five seconds. Taking the size of its grammar and the types of errors it deals with into account, IEGRED is considerably more efficient than those CALL or NLP systems we have found in the literature that handle robust language analysis and whose execution times are given (e.g. French Grammar Analyser (Barchan, et al., 1986) and MURPHY (Selfridge, 1986)). Our experiments further show that when the size of the knowledge base (the template bank for IEGRED and the phrase structure bank for the chart parser) is large and/or the input sentence is relatively long (more than six or seven words), IEGRED requires much less time (a half to one fifth dependent on the size of the knowledge base or the length of the sentence) to analyse a sentence than the chart parser (modified from Gazdar & Mellish, 1989, with improved efficiency) spends on parsing the sentence only. This is because the increase of one rule in the phrase structure bank or one word in a sentence often results in the increase of several times of backtracking during chart parsing, whereas in IEGRED, it requires only a small amount of extra time to deal with the new template or word.

Summary

Our research on robust language analysis in the context of computer-assisted language learning is described. In order to attack the difficult task of handling ill-formedness especially ill-structured sentences at purely syntactic level without error anticipation, and to address the often-ignored issue of carefully designing the language analysis method to facilitate the student's remedy work, we propose a novel architecture incorporating several AI techniques; model-based representation and reasoning, explanation-based learning, and flexible pattern matching. A new kind of knowledge representation formalism, extended place/transition tree, which is particularly suitable for automated language diagnosis for the purpose of language learning is suggested. It appears this approach has validity, since the prototype based on the phrase structure expansion architecture and the EPTT formalism, IEGRED, is indeed robust. This robustness derives from its embodiment of both flexible pattern matching and sentence structure modelling. IEGRED

performs particularly well on detecting arbitrary and multiple structural errors, in which other CALL systems are often weak (e.g. Barchan & Wusteman, 1988; Schwind, 1988).

The PSE architecture is not without its own limitations. As Menzel (1988) correctly points out, "deep modeling" can only be applied to domains in which the correctness conditions of the knowledge involved could be identified clearly and completely. This means at least at the current stage, the area in which this approach appears appropriate will be only morphological and syntactic error diagnosis. Without access to any semantic information, it is not always possible for a system to resolve structural ambiguity, for example, to distinguish the different structures of "I ate dinner with a friend." (Charniak & McDermott, 1985, 171) and "I ate dinner with a fork.". We hope that the more we understand natural language, the more applications we may find in which model-based approach will be effective.

Further work will concentrate on several specific areas. First is the improvement of the best-fit selection method. With more sentence patterns in the bank, the efficiency of the best-fit selection becomes more critical to the overall performance of this approach. The current method can handle a small database well, we have to see what will happen if the database grows. Secondly, we shall look at the possibility of incorporating semantic processing into IEGRED. At the moment, we can think of the use of some semantic markers and selectional restrictions. Although crude, semantic marker analysis can be used to reject a syntactic analysis without appropriate semantic reading, therefore help sentence structure disambiguation.

References

Barchan, J., Woodmansee, B. & Yazdani, M (1986). A Prolog-based Tool For French Grammar Analysers. *Instructional Science, 14*, 21-48.

Barchan, J & Wusteman, J (1988). A Prolog-Based Tool for Grammar Analysis of Western European Languages. Research Paper W171, Department of Computer Science, University of Exeter, England.

Brown, J. S., Burton, R. R. & DeKleer, J. (1982). Pedagogical, natural language and knowledge engineering techniques in SOPHIE I II and III. In D. Sleeman & J. S. Brown (Eds) *Intelligent Tutoring Systems*, Academic Press, London.

Burton, R. R. & Brown, J. S. (1982). An investigation of computer coaching for informal learning activities. In D. Sleeman & J. S. Brown (Eds) *Intelligent Tutoring Systems*, Academic Press, London.

Charniak, E. & McDermott, D. (1985). *Introduction to Artificial Intelligence*, Addison-Wesley, Sydney.

Cohen, V. B. (1983). Criteria for the evaluation of microcomputer courseware. *Educational Technology, 23*, 9-11.

Cohen, V. B. (1985). A reexamination of feedback in computer-based instruction: Implications for instructional design. *Educational Technology, 25*, 33-36.

Gazdar, G. & Mellish, C. (1989). *Natural Language Processing in LISP*, Addison-Wesley Publishing Company, Sydney.

George, H. V. (1972). *Common Errors in Language Learning*, Newbury House Publishers, Inc. Massachusetts.

Granger, R. H. (1983). The NOMAD System: Expection-Based Detection and Correction of Errors during Understanding of Syntactically and Semantically Ill-Formed Text. *American Journal of Computational Linguistics, 9* (3/4), 188-196.

Hayes, P. J. & Mouradian, G. V. (1981). Flexible Parsing. *American Journal of Computational Linguistics, 7* (4), 232-242.

Hunt, J. W. & Szymanski, T. G. (1977). A fast algorithm for computing longest common substrings. *Communication of ACM, 20* (5), 350-353.

Johnson, F. C. (1973). *English as a Second Language - An Individualized Approach*, John Murray, London.
Kay, M. (1976). Experiments with a powerful parser. *American Journal of Computational Linguistics*, microfiche no. 43.
Kulhavy, R. W. (1977). Feedback in written instruction. *Review of Educational Research, 47*, 211-232.
Kwasny, S. C. & Sondheimer, N. K. (1981). Relaxation Techniques for Parsing Grammatically Ill-formed Input in Natural Language Understanding Systems. *American Journal of Computational Linguistics, 7* (2), 99-108.
Leech, G. (1987). General introduction. In Garside, R., Leech, G. & Sampson, G. (Eds) *The Computational Analysis of English, a corpus-based approach*, Longman, New York, 1-15.
Mellish, C. S. (1989). Some Chart-Based Techniques for Parsing Ill-Formed Input. *Proceedings of the 27th Annual Meeting of the Association for Computational Linguistics*, Vancouver, 102-109.
Menzel, W. (1988). Error diagnosing and selection in a training system for second language learning. *COLING-88*, 414-419.
Minton, S. (1990). Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence, 42*, 1990.
Minton, S., Carbonell, J., Knoblock, C., Kuokka, D., Etzioni, O. & Gil, Y. (1989). Explanation-based learning: a problem solving perspective. *Artificial Intelligence, 40*, 63-118.
Nyns, R. R. (1990). An expert system in computer assisted language learning. *Computers Education, 15*, 99-103.
Pascoe, G. A. (1986). Elements of object-oriented programming. *BYTE*. August, 1986, 139-144.
Pullum, G. K. & Gazdar, G. (1982). Natural Language and Context-free Language. *Linguistics and Philosophy, 4*, 471-504.
Reiser, B. J., Anderson, J. R. & Farrell, R. G. (1985). Dynamic Student Modelling in an Intelligent Tutor for Lisp Programming. *Proceedings of the Ninth International Joint Conference on Artificial Intelligence*, 1, Los Angeles, California, August, 1985, 8-14.
Samuelsson, C. & Rayner, M. (1991). Quantitative Evaluation of Explanation-Based Learning as an Optimization Tool for a Large-Scale Natural Language System. *Proceedings of the Twelfth International Conference on Artificial Intelligence*, Sydney, 2, 609-615.
Schwind, C. (1988). Sensitive parsing: error analysis and explanation in an intelligent language tutoring system. *COLING-88*, 608-613.
Selfridge, M. (1986). Integrated Processing Produces Robust Understanding. *Computational Linguistics, 12* (2), 89-106.
Tadepalli, P. (1991). A formalization of explanation-based macro-operator learning. *Proceedings of the Twelfth International Conference on Artificial Intelligence*. Sydney, 2, 616-622.
Takahashi, H., Itoh, N., Amano, T. & Yamashita, A. (1990). A spellling correction method and its application to an OCR system. *Pattern Recognition, 23* (3/4), 363-377.
Weischedel, R. M., Voge, W. M. & James, M. (1978). An Artificial Intelligence Approach to Language Instruction. *Artificial Intelligence, 10*, 225-240.
Weischedel, R. M. & Black, J. E. (1980). Responding Intelligently to Unparsable Inputs. *American Journal of Computational Linguistics, 6*, 97-109.
Weischedel, R. M. & Sondheimer, N. K. (1983). Meta-rules as a basis for processing ill-formed input. *American Journal of Computational Linguistics, 9* (3/4), 161-177.
Woods, W. A. (1970). Transition network grammars for natural language analysis. *Communications of the ACM, 13*, 591-606.

Appendix

This appendix gives two sample system-student interactions. IEGRED has been developed on Macintosh with menu/window interface. Hence, the following two are much simplified running

sessions in order to save space. In a real situation, the systen's messages appear in different windows. The student's activities are described in the parentheses. The student's input is in bold. The plain text after the prompt gives the display of the system.

The First Sample Session (diagnosis mode)

(select the menu *Analyse Sentence*, type in the following sentence in a dialog box)
> **she is australian actor?**
> There are some errors in the sentence. By choosing appropriate menus, you can either try to identify those errors yourself with the assistance of the system or view the diagnostic report directly.

(select *Report/Explanation About Structure*)
> The correct sentence structure suitable for your sentence and the analysis of the structure of your sentence are shown below:

"she"	-->	a proper noun
"is"	-->	a link verb
a determiner	-->	COMPONENT MISSING
"australian"	-->	an adjective
"actor"	-->	a noun
"?"	-->	a punctuation

(select *Discover Error/Get General Error Info*, choose "australian actor" from a list of sentence parts)
> The sentence part "australian actor" contains the following error(s):
1. incorrect non-capitalized word.

(select *Discover Error/Get Specific Error Info*, choose "she is australian actor")
> The sentence part "she is australian actor" contains the following syntactic concord error(s):
1. The words "she" and "actor" have different gender;
2. The first word in the sentence, "she" here, was not capitalized;
3. The adjective related to a proper noun, "australian" here, was not capitalized.

(select *Discover Error/Get Specific Error Info*, chooses "?")
> The sentence part "?" contains the following syntactic concord error(s):
1. The final punctuation of the declarative sentence should have been ".", not "?".

The Second Sample Session (drill mode)

(select *Enter Drill Session/Drill/Select Drill Pace/Fast*)
> The current drill pace has been set to FAST.

(select *Start Drill*)
> In an exercise in this drill, you are required to write a grammatically correct sentence according to the structure of the given example and the key words. For instance, given the following information

Ex: He is tall.
they/strong --> They are strong.
Key words: I/happy

you may write a sentence such as "I am very happy today." as a correct answer.

(click the OK button in the dialog box)
> Ex: The bottle is on the table.

boys/room --> The boys are in the room.
Key words: books/shelf
Enter an appropriate sentence: (an editable-text box appears here)

(enter the following sentence)
> **The books are in the desk.**
> The word "books" is not in the built-in dictionary. IEGRED guesses it is the plural form of the following noun:
book
Do you accept it?

(click the YES button)
> There are some errors in the sentence. By choosing appropriate menus, you can either try to identify those errors yourself with the assistance of the system or view the diagnostic report directly.

(select *Get General Error Info*; choose "in the desk")
> The sentence part "in the desk" contains the following error(s):
1. incorrect collocation between a preposition and a noun
2. missing key word(s)

(select *Get Specific Error Info*, choose "in the desk")
> The sentence part "in the desk" contains the following syntactic concord error(s):
1. The collocation of the preposition "in" and the noun "desk" is inappropriate as far as the given context is concerned;
2. A given key word was not used in the input sentence.

For further explanation as to the above error(s), please select the menu Explanation For Wrong Components under the menu Report.

(select *Explanation For Wrong Components*, choose "1" from a list of error entry numbers)
> In English, certain nouns are often associated with certain prepositions to form preposition phrases. This is called the COLLOCATION of preposition and noun. Preposition phrases are very useful in English. However, since there is no a general rule for constructing preposition phrases, we have to remember a noun and the associated preposition as a whole in most of the cases.