DOCUMENT RESUME

ED 345 524 FL 020 219

AUTHOR Helmreich, Stephen C.
TITLE Template Construction as a Basis for Error-Analysis Packages in Language Learning Programs.
PUB DATE 87
NOTE 12p.; For the journal as a whole, see FL 020 212.
PUB TYPE Reports - Descriptive (141) -- Journal Articles (080)
JOURNAL CIT IDEAL; v2 spec iss p97-106 1987

EDRS PRICE MF01/PC01 Plus Postage.
DESCRIPTORS *Authoring Aids (Programing); Computer Assisted Instruction; *Computer Software; *Error Analysis (Language); *Pattern Drills (Language); Programing; Second Language Instruction; *Second Languages; Sentence Structure

ABSTRACT

An "intelligent" system for constructing computer-assisted pattern drills to be used in second language instruction is proposed. First, some of the difficulties in designing intelligent error analysis are discussed briefly. Two major approaches to error analysis in computer-assisted instruction, pattern matching and parsing, are described, and their advantages and disadvantages are examined. A third approach that appears to minimize technical problems is then outlined. This approach is based on the idea that a limited list of randomly generated sentences in the target language is useful in constructing transformation or pattern drills. The system consists of four parts including: (1) a module in which sentence templates can be constructed; (2) a procedure for transforming linguistic material into target language sentences; (3) stimulus sentence creation, including student response; and (4) error analysis, providing feedback to both the student and the sentence-generating system and allowing tailoring of further stimulus sentences. Some problems of implementation are discussed, including system response speed, feasibility of expansion beyond simple sentences, generation of inappropriate sentences, and constraints imposed by left-to-right template construction, transformation, and response judging. (MSE)

Template Construction as a Basis for Error-Analysis Packages in Language Learning Programs

Stephen C. Helmreich

IDEAL 2, 1987

TEMPLATE CONSTRUCTION AS A BASIS FOR ERROR-ANALYSIS PACKAGES IN LANGUAGE LEARNING PROGRAMS

Stephen C. Helmreich

In this paper, I describe a set of programs which work interactively with the teacher/instructor to create a template from which an indefinite number of stimulus sentences can be generated. The instructor also specifies exactly how the stimulus sentence is to be transformed into the target sentence. Using this set of transformation steps, the student's response is judged and feedback is supplied to the student. Feedback is also given to the stimulus-generating system, allowing it to tailor the construction of further stimulus sentences to the difficulties encountered by the student in previous attempts.

INTRODUCTION

Computerization of language learning lessons is still in its early stages. In particular, the analysis of student responses to drill items is generally simplistic and the feedback often counter-productive. In this paper I describe a computer program that I believe represents a substantial advance over systems currently available. It permits pedagogically oriented, drill specific feedback and alters the presentation of stimulus material in response to errors encountered in analysis of previous student responses.

Error analysis is the main focus of the approach I am proposing here. It is in this area that the program offers the most substantial improvement over other approaches. However, the program itself encompasses more than just an error analysis package. It offers, in fact, a complete computerization of a certain type of language drill, namely, transformation exercises, or pattern drills. Once it has been provided by the instructor with a pattern for the drill and a procedure for deriving the correct response from the stimulus, the program will construct appropriate stimulus sentences, correctly determine the target sentences, analyze the student's response, and provide feedback to both student and teacher. It may also handle a small group of highly constrained translation exercises. However, by itself, it cannot provide a complete set of computerized language-learning lessons.

In the following section I provide background information, pointing out some of the difficulties of intelligent error analysis, describing alternative approaches, and placing template construction within the larger framework of language generating systems. Subsequent sections provide a detailed description of the system and a conclusion, which summarizes both the advantages and some disadvantages of this approach.

BACKGROUND

Problems of Intelligent Error Analysis

We as teachers are continually involved in error detection, error correction, and error analysis. These tasks are performed routinely and are well within the capabilities of most teachers. One might think, therefore, that computerization of this process would be fairly simple. However, what we are doing is something extremely complex and subtle. In analyzing errors, we are trying to ascertain the invisible, internal cognitive processes which produced the error. The identification of these processes is vital to the task of providing appropriate feedback to the student to aid in correction of the error. Moreover, these

processes are not even open to conscious introspection, since it is usually the student's conscious intention to produce the correct answer. However, the main difficulty in intelligent error analysis is that the interpretation of the objective error is ambiguous.

First of all, it is ambiguous linguistically. The same physical error may represent a cognitive error at several linguistic levels. It may be an error at a fairly low level (typing, spelling), at an intermediate level (inflection, agreement, word order), or at a high level (word choice, idiomaticity, appropriateness). For instance, finding the word "chevaux" instead of "cheveux" in a French sentence may be a spelling error, a morphology error, or possibly a word choice error.

Second, there are many possible causes for any particular error. For instance, one and the same error may represent a simple oversight, a temporary lapse, a gap in the student's knowledge, interference from the student's native tongue, a misunderstanding of the instructions, or a major misconception about the target language. Our choice often depends on many other things we know about the student: general level of knowledge, past performance, etc.

Another factor which makes the computerization of error analysis difficult is that it is task dependent. That is, an error in one situation may not be an error in another. For instance, vocabulary tests are generally geared to a particular set of items. Using another word with the same meaning might get the student marks for ingenuity, but would still be a wrong answer. Even if errors "count" in different situations, they may have different weights. For instance, a spelling error would count heavily on a spelling test, and probably for little in an extended essay.

Thus, what makes intelligent error analysis difficult to computerize is the fact that physical errors in the text are ambiguous. Their interpretation and evaluation depends on an analysis of the linguistic import of the error and of its probable cause. Even if this can be done, the weight given to errors varies from exercise to exercise.

Other Approaches to Computerized Error Analysis

There have been two major approaches to error analysis in computer-assisted learning programs. The first is simple pattern matching. With this approach, the student's input is treated as a flat (unstructured) string of characters. The input is searched for certain words or phrases. This method requires a pattern (a correct answer) against which the student's input can be matched. As such it is a fairly unintelligent approach to error analysis. However, in its favor, it can serve as a general purpose approach, workable for almost any input in any language.

Parsing the input is a more intelligent approach to error analysis. Because of work done both in computer science and linguistics on the nature of grammars and parsers for natural language, some of the principles of parsing and some of the basic structures of natural languages are fairly well understood. The parsing approach, therefore, tries to analyze the student's input in structural terms, not simply as a flat string of characters. The structures it looks for are those of the foreign language, so it must "know" something about these structures. In this sense, it is more intelligent. In addition, it is not bound to situations where a clear-cut pattern is available against which to judge the student's input. With an adequate grammar of the target language, a parser can handle almost any input in the language. In this respect, it appears to mimic human analysis of language. Since we do not always know what it is we will be hearing or reading, we must analyze the input we receive and assign it a structure and meaning.

It would be tempting to think that this work could be taken over directly into error-analysis programs. However, almost all such work deals only with analyzing correct input. Bad or incorrect input is generally discarded or ignored. Incorrect input, unfortunately, is exactly what must be examined and analyzed in error analysis. In addition, a parser that can analyze most natural language input and operate at a reasonable speed will be quite complex and unintuitive. This is because the tasks of computationally optimizing a parser and of providing a linguistically sophisticated grammar are both highly complex and technical. This means that it is difficult to make such programs interactive and task dependent. It also is not an easy job to take the complex analysis provided by the parser and turn it into helpful

and non-technical feedback to the student and teacher. "Violation of Binding Principle B" is not always helpful feedback to a student trying to learn reflexives.

I do not want to suggest that the parsing approach should not be pursued in the development of computer assisted language learning programs. In fact, it is clearly necessary if these programs are to deal with free input from the student. However, as outlined above, there are several large problems that must be solved in the process of applying what we know about language structure in general, and computerized parsing routines in particular, to computerized language-learning programs. Therefore, I am suggesting another approach that does not seem to pose quite so many technical problems. Although it is applicable to a more limited domain than the parsing approach, it may prove quite useful as an intermediate stage in the development of "intelligent" computer programs. This approach involves the use of templates or generating systems.

Generating Systems and Templates

From the standpoint of generative linguistics, a generating system, or language generator, is simply the inverse of a parser. That is, a generative grammar is a formal system that associates each "phonetic form" (sentence) of a natural language with a "logical form" (meaning). It is similar to a mathematical function in that it is simply a set of ordered pairs, <sentence, meaning>. A parser is also a function in the less technical sense of a procedure which can be applied to an input (namely, a sentence of a language) and produces as output a structured meaning for the sentence. A generating system is simply the inverse function or procedure. That is, it takes as input a meaning and produces as output a sentence of a natural language that encodes that meaning.

It is easy to see why parsers have been a more popular area of research than generating systems. First of all, there is disagreement about what a "meaning" is or looks like. By contrast, a "sentence" is a fairly objective thing. So it is certainly more feasible to start with a sentence and work toward some sort of logical/semantic representation than vice versa. Second, in most cases, the practical goal toward which researchers are aiming is a program that can "understand" natural language input. Generally, computers have very little that they need to communicate, and this can be achieved with canned responses and simplified statements that do not require sophisticated processing. Therefore, a parser would be much more useful than a generating system.

Seen from this point of view, generating systems pose even more difficult problems than parsers, because there is no clear consensus on what the input (a meaning) should look like. The problems of adapting such a system to the analysis of incorrect student input seem even more formidable than for a parser. And the complexity of the program would certainly rival that of parsers.

There is another type of program that is generally referred to as a language generator. Given a formal grammar, with phrase structure rules (and possibly transformations), this program randomly generates grammatical sentences of the language defined by that grammar. This type of generator is completely divorced from any semantic considerations. It is fairly simple to implement technically, but its usefulness in error analysis systems is unclear. How could a list of grammatical sentences, randomly generated, help in analyzing student errors?

The approach described here, which I call a "template approach," falls somewhere in between these two technical types of generating systems. It is grounded in the fact that a constrained list of randomly generated sentences would be useful in the construction of a particular type of language-learning exercise, namely the transformation drill.

In this type of exercise, the student is presented with a set of sentences and told to perform some specified action to each of them. For instance, typical transformational drills include pluralizing noun phrases, changing the tense of sentences, turning statements into imperatives or questions, or replacing a noun phrase with an appropriate pronoun. In most cases, the meaning of the sentence is not of paramount importance. Although the sentences should not be semantically anomalous, the purpose of the drill is to inculcate certain syntactic transformations. The stimulus sentences are generally similar in syntactic form,

and the student response is usually obtained by a straightforward syntactic manipulation of the stimulus sentence.

What I am suggesting then is a program which would allow the teacher to specify a syntactic template (using terminology of a non-technical nature, such as that found in most pedagogical grammars) for the generation of stimulus sentences. The teacher then specifies exactly how this stimulus sentence is to be transformed into the response sentence. Using this information, the program can generate any number of stimulus sentences and also automatically produce the correct response. Then, since it has an understanding of the steps to be followed to create the correct answer, it can easily check the student's response to see if those steps were followed. A program like this might be viewed as a rough, non-technical application of early transformational grammar to the construction of language-learning drills.

If this were all the program did, it might be a time-saver for the teacher, that is, if constructing the template and the transformation took less time than typing in the stimulus sentences themselves. However, using this approach, informative and relevant feedback can also be provided to the student by the teacher. This feedback is geared to the detection of errors at each step in the transformation of the stimulus sentence into the target sentence. In addition, by gathering information about the errors committed by the student in the course of a drill, immediate feedback to the generating system results in the generation of more sentences of the type which posed some problems for the student.

DESCRIPTION OF THE SYSTEM

In this section, I will explain in more detail the construction and operation of the proposed program. A prototype model of most of the system has been written, though with minimal implementation of each section and with no regard for user-friendliness. In order to facilitate an understanding of how the system works, I will also describe the functioning of the program in the construction of a particular transformation drill. This is a simple French pattern drill requiring the student to pluralize simple sentences in the present tense. For example, the stimulus sentences in (1) should prompt the responses in (2).

(1) Je vais au cinéma.
Le chat mange le poisson.

(2) Nous allons au cinéma.
Les chats mangent le poisson.

As can be seen in Figure 1, the system consists of four major parts: the template construction module, the transformation procedure module, the stimulus creation program, and the error analysis program. In addition there are some built-in, language-specific functions that provide short-cuts in the construction of templates and transformations. There is also a large "reverse lexicon." These will be discussed at the appropriate place as they are called upon by the four main programs.

Template Construction

Template construction can be visualized as the creation of a tree structure which reflects the various syntactic shapes that a stimulus sentence can take. Figure 2 shows a possible template for the construction of our example sentences. The tree is constructed in a top-down, nearly depth-first manner. Each node (branch-point) of the tree is named, generally with a mnemonic code representing the syntactic category of the leaves (terminal nodes) of the tree. For instance, the root node of the template constructed for our example might be called SENT, since the template will create sentences. Naming each node is important, since they can then be referred to when describing the transformation necessary to produce the target output.

The branches under each node must be one of three possible types: they may lead to options in stimulus construction, they may lead to required parts of the stimulus, or they may

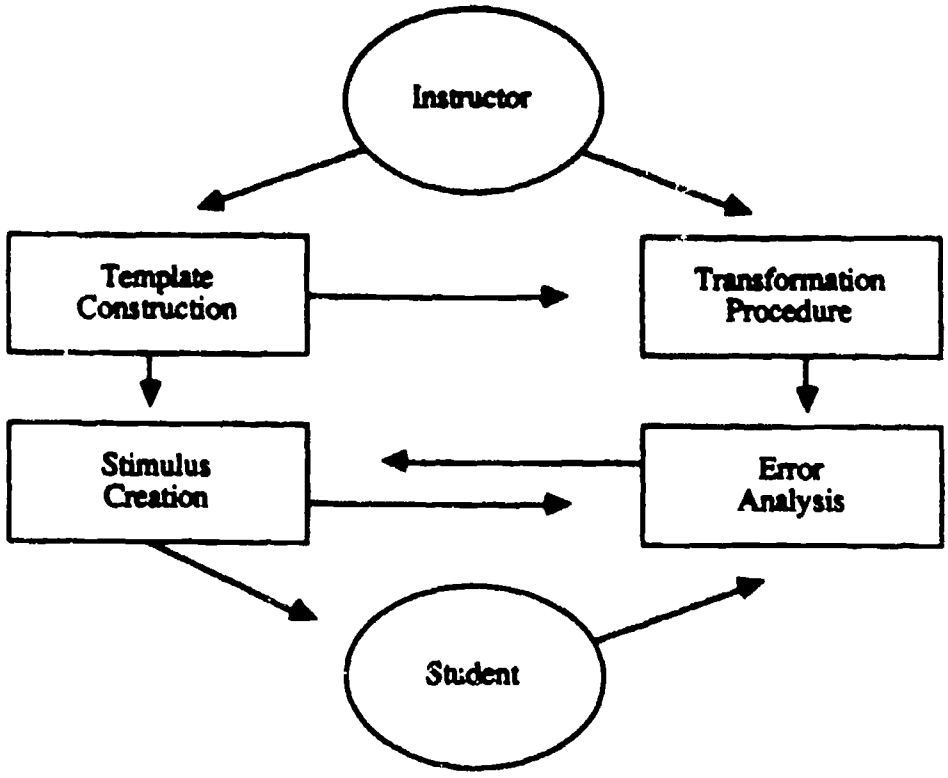

Figure 1. Model of a template construction system for error analysis

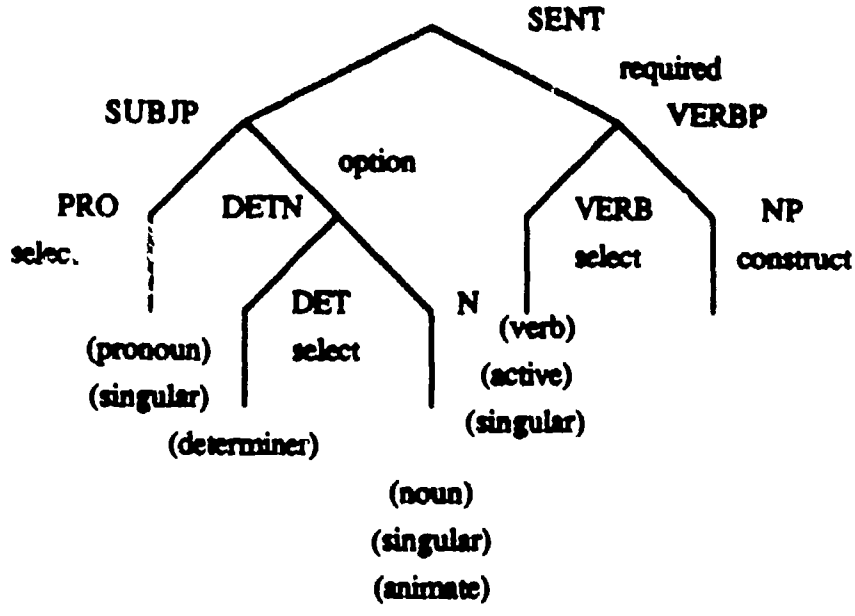

Figure 2. Template construction

lead to the selection of a lexical item to fill a spot in the stimulus sentence. The instructor must specify which type of branching occurs beneath each node. For instance, in our example, beneath the node SENT, the instructor specifies two other nodes, SUBJ and VERBP. These will subsume the subject and the verb phrase of each sentence. Since both a subject and a verb phrase are required for each sentence, each branch must be taken when constructing a stimulus sentence.

However, under the node labelled SUBJ, we find nodes labelled PRO and DETN. These nodes represent the construction of a subject consisting of either a pronoun or a determiner noun sequence. Since either is a complete subject and both together would be ungrammatical, the branches to these nodes are options for the stimulus constructing program--it must take one and only one of these paths.

Finally, a branch may lead to a terminal node (leaf) of the tree. In this case, a lexical item must be selected to fill the terminal node. The instructor is asked to specify the lexical category and the features that the lexical item must bear. In this case, the features "singular" and "animate" have been selected. In the PRO branch the lexical category "pronoun" has been selected and in the DETN branch the lexical category "common_noun." When a lexical item is to be selected, the instructor must also specify whether the lexical item selected must agree with any other item in the template. In our example, the determiner must agree with the subject noun and the subject noun with the verb. Both the feature system and morphology will be discussed in detail below.

In addition to the completely teacher-specified templates, there are additional built-in functions which can generate random constituents such as noun phrases, prepositional phrases, verb phrases, adverbial phrases, etc. If the content of these constituents is not relevant to the exercise at hand, the teacher need not specify completely their construction. For instance, in our example, the form and content of the object noun phrase is not relevant to the task of pluralizing the sentence.

Features

Features play an important part in this template system, since it is by means of features that much of the flexibility of the system is obtained. Most lexicons are organized as shown in (3), with each lexical item attached to a list of its own features. In this system, however, the feature names are entered with a list of the lexical items bearing that feature attached, as shown in (4).

(3)	chat	(noun masc animate common)
	plume	(noun fem common)
	manger	(verb active)
(4)	noun	(chat plume)
	common	(chat plume)
	verb	(manger)

This organization means that the selection procedure involves taking the sets associated with the features selected, intersecting them, and randomly choosing one lexical item from the intersection. This item will bear all the features selected by the teacher.

It is important to note that neither the lexicon nor the set of features is static. Words can be easily added and features can be easily assigned. This means that the selection process for lexical items can be closely controlled by the teacher. For instance, a feature could be created such as "noun from the vocabulary list for lesson nine." Using this feature, the teacher could limit selection to certain vocabulary items. The features could also be semantic in nature. A possible feature might be "articles of clothing" or "things to eat." Thus by the use of this flexible feature system, semantic constraints can be ensured.

It is also important to note that these features, whether syntactic or semantic, are intended to be inherent features of the lexical item itself. Morphological variation (primarily the agreement features) would be taken care of by independent pre-programmed modules. Therefore, to enter a regular lexical item it would not be necessary to enter more than the

citation form. Morphological variation would be handled automatically. (Irregular variation would need to be entered into the appropriate computation module.) These functions, when provided with a lexical item and a set of morphological features, return the correct morphological form as their result. If a feature is left unspecified (say, for example, tense), it would be supplied at random. In addition, this functional approach to morphology allows closely related and even incorrect forms (a regular stem in place of an irregular correct stem, or a future stem with a present ending) to be generated and stored with the correctly generated form. These are used later to perform a primitive sort of morphological error checking.

Transformation Procedure

Once the template is constructed, the teacher must then specify exactly what actions are to be performed in order to create the correct target sentence. The transformation program operates from left to right, so changes must be specified in sequence through the sentence. For instance, in our example, the instructor must specify first changes to be made to the subject noun phrase and then to the verb phrase.

Any node of the stimulus template may be specified as subject to transformation. The instructor has four options to choose from in specifying the appropriate transformation of the stimulus sentence: copy, insert, delete, or alter. **Copy** results in a verbatim copy of whatever lexical items lie under the node selected. **Insert** involves the insertion of specified material. New material may be specified, or another node from a different part of the stimulus. Insertion must occur **between** constituents of the stimulus sentence. **Delete** is simply the removal of the specified material. Thus, **insert** and **delete** together provide a movement transformation. **Alter** involves changing the feature matrix of a specified lexical item, such as tense, number, and other syntactic/morphological features. In addition, there are available several pre-programmed functions which can provide certain commonly-performed feature alterations. These would be available to apply to larger scale constituents than lexical items. Such functions as changing tense, number, or case would be available, allowing the teacher to avoid directly manipulating the feature values themselves.

After specifying the constituent and the action to be performed on it, the teacher may provide an error message to be used if the student fails to execute properly this part of the transformation. Default messages for each kind of transformation are also available.

As can be seen from the above specifications of the stimulus template and the transformation construction, there is more than one way in which the same drill could be formulated. The instructor can choose how detailed a template to construct. The more detailed a template, the more detailed can be the specification of the transformation. The more steps in the transformation, the greater the number of "errors" the student may commit--each "error" being related to one step of the transformation. Thus the teacher controls how detailed the analysis of the student's work is.

Stimulus Creation

Given the template structures for a drill as specified by the teacher in the foregoing sections, the actual creation, presentation, and judging of a transformation exercise is fairly routine.

First, using the template, a stimulus sentence is constructed. That is, starting at the root node, the program traverses the stimulus template tree. If the node it is examining is has optional branches, only one branch (randomly chosen) will be traversed and the others ignored. If the branches are specified as part of a construction, each branch will be traversed in turn. If the node is a terminal node, an appropriate lexical item is selected. Agreement routines are executed. The tree-structure for the stimulus is stored and the lexical items are strung together and presented on the screen to the student.

Once the student has typed in a response, the judging sequence begins. Following the transformation schema specified by the teacher, the program checks each step to see that it has been successfully completed by the student. Thus, a copy step merely involves checking to see if the identical material in the stimulus sentence is in the student's response, and at the

appropriate place. Insertion and alteration steps check f r the appropriate inserted or altered information. In addition, if an error is found in an alteration step, morphologically similar forms generated earlier are checked to see if the student's response is among them.

Deletion of material cannot be directly checked since there is no way to see if something is not there, so evaluation is deferred. If the following step succeeds, it is presumed that the prior deletion was properly executed. If not, the program checks to see if the deleted material is still present in the student's response.

In each situation, if evaluation fails, the appropriate error message is printed. Thus, the student is led through the necessary transformations step by step.

Error Analysis

In addition to providing appropriate response to student errors, the program allows for a feedback loop from error analysis to stimulus generation, since stimulus sentences are spontaneously generated. This means that the stimulus sentences the student is presented with can be weighted to favor those types of sentences that proved difficult in earlier attempts.

For instance, data about which branch of the option node of a template was chosen is stored. Information about the number of errors a student made for each constitutent of each sentence is also correlated with the branches present in that sentence. This allows for the weighting of option nodes, so that options which resulted in sentences causing difficulty appear more frequently.

I want to emphasize the importance of this feedback loop, since it represents one of the significant advantages of this approach over a parser-based approach. Given an adequate parsing and error-analysis routine, a teacher would need only to type in the stimulus sentences, and the correct response. The parser could easily parse the correct response and use that as a target against which the student response would be judged. However, such a program could not generate new stimulus sentences, and thus could not individualize the drill appropriately for each student.

CONCLUSION

The "Intelligence" of the System

At this point, it is appropriate to summarize and highlight those aspects of the proposed approach which permit it to be somewhat "intelligent" in the creation of transformation drills and analysis of student responses.

In the first place, the teacher interacts with the program to create the drill. The teacher's intelligence is, therefore, integrated into the construction of the drill. This is made possible by several features of the program. First, the flexibility of the template construction process allows the teacher to specify in greater or lesser detail the structure of the drill template. Second, the ease with which features may be added to the system permits the teacher to tightly constrain the random choice of lexical items for the drill. Third, the allowance in the program for extended, specific error messages written by the teacher incorporates the teacher's knowledge of probable student errors.

In the second place, there is the built-in knowledge of the language that allows for "intelligent" drill construction. This knowledge includes the built-in lexicon and feature system, the construction functions which contain information about the grammar of certain constituents of the language, and the morphological functions which contain stored information about the regularities and irregularities of morphological systems in the language. This built-in knowledge interacts with the teacher's input to allow the instructor to concentrate on specifying the significant constructions for the purpose of the exercise and allows peripheral areas to be handled automatically, with minimal attention.

Finally, there is the feedback loop from error analysis programs to the construction of stimulus sentences for the drill. This allows "intelligent" construction of stimulus sentences.

That is, sentences are presented that help the student learn by concentrating on areas where the greatest difficulty lies.

Problems of Implementation

A few problems that may arise in the complete implementation of this system should be mentioned. These are mainly problems of scale. That is, there are questions of how well the system would work with a large lexicon, a large set of features, and a significant set of built-in functions.

The first concern is whether or not the response speed of the system can be maintained with such a larger data base. I suspect that a reasonable speed can be maintained, particularly if compiled code is used.

The second concern is whether or not the set of features and constructions necessary to handle a wide variety of constructions in the target language would make the system too unwieldy for the average teacher to work with. As with most systems, unforeseen problems will arise as it expands, demanding exceptional or ad hoc solutions. This may make the system too large to be handled by a typical foreign-language teacher without a great deal of linguistic or computational background.

The third concern is about the nature of the stimulus sentences generated by the program. There is currently a great deal of interest in communicative competence and the simulation of appropriate conversational situations. This program, with its limited semantic knowledge, will probably generate grammatically correct, but semantically odd sentences. How odd, it would be difficult to tell. To some extent, these odd sentences might be amusing to the student/learners. However, care must be taken that the oddness does not blend into "wrongness". It might be acceptable for the program to generate sentences like "Sally ate the stone". However, it should not generate a sentence like "Sally ate the sincerity." Avoiding sentences like the latter might overburden either the system or the teacher.

Finally, there is a concern about the current left-to-right procedure in template-construction, transformation, and response judging. A brief look at typical pattern drills shows that it would not pose unnatural constraints on the instructor to specify transformational changes in a strict left-to-right order. However, in some cases, it might be unintuitively or pedagogically better to work through the transformation in some way other than left to right.

The advantage of this strictly sequential method is that it avoids one of the major problems of a pattern matching approach. That is, in matching up the student input with the correct response pattern, it is very difficult for pattern matching to deal with misplaced constituents or scrambled word order. Of course, it also can match sequentially left to right, stopping if it fails to match exactly. But because there is no structure to the pattern, there is no way it can know more about the error than that it is not an exact match with the correct response. With the template approach, the program "knows" what each constituent in the answer should be. If it fails to match at a particular point, an appropriate error message is generated, helping the student correct the error at that point.

Summary

In this paper I propose an "intelligent" system of computerized drill construction. It operates interactively with the instructor to create a template for stimulus sentences and a sequence of transformations which change the stimulus sentence into the target sentence. Using a built-in system of features, morphological functions and constructions, the program presents stimulus sentences generated according to the template and judges the student response based on the steps in the transformation process. A feedback loop permits the generation process to be sensitive to errors committed by the student.

The positive features of such a system include the following: (1) it allows the instructor flexibility in constructing transformation drills that are specific to the instructional goals of the teacher and the learning difficulties of the student; (2) it encourages clarity on the part of the teacher in specifying clearly what the possible shapes of the stimulus

sentences are and what must be done to produce the correct responses; (3) it enables flexible (goal-related) error analysis; (4) it permits the construction of an indefinite number of drill sentences, differentially for each student in relation to prior errors.

ACKNOWLEDGMENTS

I would particularly like to thank Dr. Robert Hart for his assistance with this article. Many of the ideas underlying this approach to error analysis I owe to him.

THE AUTHOR

Stephen Helmreich is a graduate student in the Department of Linguistics at the University of Illinois. He works as a research assistant in the Language Learning Laboratory.