Language Technology and Language Instruction: Computational Diagnosis of Word Level Errors

Gerard Kempen
NICI, University of Nijmegen, Montessorilaan 3, 6525 HR Nijmegen, The Netherlands

Abstract: Three software packages for grammar and spelling instruction are described, all of them intended to run on personal computers: (1) an 'intelligent' training program which teaches how to conjugate and spell Dutch verbs; (2) a grammar curriculum which covers almost one hundred concepts and terms for describing grammatical structures and includes a training environment for applying them to sentences constructed by the students themselves; and (3) a student wordprocessor which, in addition to editing facilities, can provide help about the spelling of some 10,000 Dutch words and about the rules governing their inflection. The paper emphasizes the treatment of inflection, agreement, and spelling errors: (1) a search technique which relates manifest inflection errors to points of departure from the formal inflection algorithm; (2) a syntactic parser for detecting syntactic agreement errors; and (3) an accurate spelling checker which can deal with both typographical and orthographical errors ('look-alikes' and 'sound-alikes'). It is concluded that these techniques, which do not aim at full-fledged student modeling, can yield satisfactory results in the context of guided sentence production in first and second language instruction.

Keywords: grammar instruction, spelling instruction, syntactic parsing, error diagnosis, student modeling, guided sentence production, word processing.

1. Introduction

Computational Linguistics and Artificial Intelligence offer powerful methods for detecting, diagnosing, and correcting student errors in written first and second language (L1 and L2) production. This contribution concentrates on three types of word level errors: spelling, inflection, and agreement errors. Word selection errors, a fourth type, is addressed elsewhere in this volume [8]. In three projects we have developed techniques for the computational treatment of such student errors occurring in written compositions or in sentences which were generated under control of a tutorial system.

After brief characterizations of the three projects in Section 1, I will lay out in some detail the three error diagnostic techniques we have elaborated (Section 2). Section 3 (Discussion) introduces two general distinctions between methods of error diagnosis: *descriptive* vs. *explanatory* methods, and *single-error* vs. *multiple-error* methods. Against the background of this typology I then assess the merits of our approaches to error diagnosis in comparison with considerably more complex student modeling techniques. At the outset I wish to emphasize

NATO ASI Series. Vol. F80
Intelligent Tutoring Systems for Foreign Language Learning
Edited by M. L. Swartz and M. Yazdani

that the projects introduced in Section 1 deal with L1 instruction. However, I believe that our diagnostic techniques can easily be adapted to the requirements of L2 instruction.

2. Three Projects

The Language Technology group of NICI has developed three software packages for grammar and spelling instruction, all of them intended to run on microcomputers of the IBM PC-AT variety:

—an 'intelligent' training program which teaches how to conjugate and spell Dutch verbs
—a grammar curriculum covering almost one hundred concepts and terms for describing grammatical structure, and a training environment for applying them to sentences constructed by the the students themselves, and
—a student wordprocessor which, in addition to editing facilities, can provide help about the spelling of some 10,000 Dutch words and about the rules governing their inflection.

I will discuss these packages in turn.

2.1 Conjugation and spelling of verbs

We have followed a strictly algorithmic approach to teaching the conjugation rules which govern the synthesis of written verb forms of Dutch. The algorithm we devised can be broken down into a *morphosyntactic* and an *orthographical* stage. The former serves to select a 'formula' specifying a string of prefixes and/or suffixes to be attached to the verb's stem. An example is *'stem+d/t+en'* for regular past tense plural verbs (equivalent to English 'stem+*ed*"; the choice between *d* and *t* depends on the stem's final letter: *d* is chosen iff that letter is a vowel or a voiced consonant). The selection process involves traversing a decision tree whose nodes pose various questions concerning finiteness, tense, number, person, etc. In the latter stage this formula is converted into a character string. After determining the verb's stem and inserting it into the formula at the appropriate place, various other string manipulations are carried out, e.g., (de)gemination of vowels/consonants (as in Eng. *occurred)* and choosing between allomorphs (e.g. *-ed* vs. *-d)*. Each stage involves several steps—decisions to be taken or string modifications to be performed.

Since verb forms involving a dozen steps are quite common in Dutch, there are many occasions for systematic or incidental student errors. A special difficulty arises from the fact that, for many verbs, the conjugation rules generate strings which are orthographically distinct but homophonous. E.g., the verb forms *wend* and *wendt* (of infinitive *wenden,* to turn) both end in a voiceless /t/, but the former is first person, the latter third person singular. To make matters even worse, there also exists a third homophonous form *went,* which belongs to the paradigm of the infinitive *wennen* (to get accustomed). Another example is provided by the verb *verkleden* (to change one's clothes) which is homophonous with three forms spelled differently:

verkleden	(infinitive; present tense plural)
verklede	(past participle, inflected)
verkleedden	(past tense, plural)
verkleedde	(past tense, singular)

The two stages of the algorithm correspond to two basic layouts of the computer screen. The morphosyntactic stage is visually displayed in the form of a decision tree whose leaves represent the various formulae. At each non-terminal node a yes/no question is asked about a morphosyntactic feature of the desired verb form (which tense, person, number?). The tree is

not shown to the student in its entirety. Instead, the branch leading to a formula unfolds node-by-node in response to the student's answers. For instance, the above formula, which underlies *verkleedden,* is arrived at via the following four questions-and-answers:

Finite verb? Yes
Imperative? No
Present tense? No
Number singular? No.

The steps of the orthographical stage are displayed on successive lines of a sheet of 'scrap paper'. The first line shows the infinitive form, and at each following line the student modifies the character string which the program has automatically copied from the previous line. The operation performed at the last line yields the final verb form. The *verkleedden* example involves the following steps:

A. Determine stem of infinitive verb *verkleden*
   Take away -en: *verkled*
   Vowel gemination: *verkleed*
B. Apply formula to stem *verkleed*
   Final consonant of stem is voiced: Add *-d+en*
C. End result: *verkleedden.*

At any step during the morphosyntactic or orthographical stage, as soon as the student makes an error the program gives relevant feedback. At any time, the student can issue requests for information, and is free to shortcut the route through the algorithm by volunteering the final answer to the current exercise.

When preparing a training session with the program, the teacher may select from a menu the type(s) of exercises to be presented to the student (e.g. past tenses and past participles only). S/he can also specify the level of detail at which the student responses will be stored in a logfile. The program guides the student through a random sequence of exercises of the specified type(s). These are selected from a set of almost a thousand. (The teacher can easily add new exercises.) Each exercise consists of a sentence with one position left open, an infinitive verb to fill the blank, and a few codes to indicate the morphosyntactic features of the desired verb form (e.g. 'inflected form of past participle'). The latter are not revealed to the student but serve the program to compute its own answer. The student is requested to reconstruct the morphosyntactic features from the sentence and to type the correctly conjugated form. The program then computes the correct form by applying the inflection algorithm to the infinitive. Whenever its own answer deviates from the student's, it starts up the diagnostic procedure to be discussed below. Usually this results in a proposal to the student to backtrack to a specific step of the algorithm. The program creates the corresponding screen layouts and requests the student to start over from there. For more detailed descriptions of this program and its motivations see [3, 9].

So far we have tested the program on L1 learners only, i.e., on native speakers of Dutch who wish to improve their mastery of Dutch spelling. However, the conjugation algorithm embodied in the program is fully explicit and presupposes no implicit morphological or phonological knowledge of Dutch on the part of the student. This makes it a relatively easy job to adapt the program to the needs of L2 learners of Dutch. We have also begun to explore possibilities of extending the approach to other languages, in particular to German, French and Latin, and are optimistic about the outcome.

2.2 A grammar curriculum

Schools in The Netherlands and in Belgium pay much attention to grammar and spelling instruction. This is due to the rule-based nature of Dutch spelling as exemplified in Section 1.1

on one hand, and to the great importance these countries attach to foreign language skills on the other. We have developed a computer-assisted curriculum which introduces and explains virtually the complete grammatical terminology needed to describe the grammatical structure of non-compound sentences. The grammar theory at the background is Incremental Procedural Grammar (IPG; see [7]), a predecessor of Segment Grammar [4, 5]. The notation and the representation of sentence structure have been simplified and brought in line with pedagogical grammars as typically taught in Dutch and Flemish schools. Sentence structures can be graphically displayed in the form of trees, at a level of detail which is in keeping with the student's current knowledge (see Figure 1).

An important innovative feature of the program concerns the so-called *constructive* exercises it offers. Students familiarize themselves with the grammar terminology and learn to apply them in the course of constructing new sentences out of menus of words or word groups displayed on the screen[1].

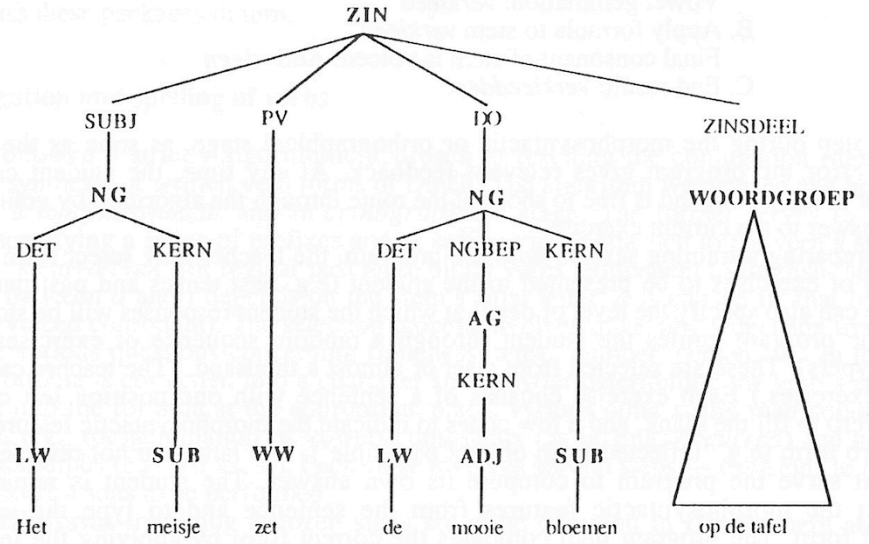

Figure 1. Syntactic tree for a Dutch sentence whose word-by-word translation into English reads *The girl puts the beautiful flowers on the table.* Some non-obvious abbreviations of grammatical terminology: zin=sentence; pv=finite verb; do=direct object; zinsdeel ('sentence part')=constituent; ng=noun phrase; woordgroep=word group; kern=head; ngbep=nominal modifier; ag=adjective group; lw=article; sub=noun; ww=verb. The triangle 'hides' grammatical concepts which have not yet been introduced to the student (prepositional phrases).

These sentences are analyzed by an IPG parser, and the program checks whether the resulting parse trees comply with the current task instruction. The parser is the core of the error diagnosis. It is geared to the evaluation of student responses elicited in the sentence construction exercises, especially errors of syntactic agreement. We will discuss its design in Section 2. For further details of the program, I refer the reader to [9].

[1] For another type of constructive exercise, see Zock (this volume).

2.3 A student wordprocessor

In order to facilitate the transfer of grammar and spelling training to writing skills, we are building a text editor which supports authors whose mastery of the morphology and orthography of the target language is incomplete. While writing a paper, letter or essay, students can activate various help facilities relevant to a selected word of the text, e.g., provide information about the inflection paradigm and the rules governing it, or about the existence of homophonous lexical entries and their meanings. If the student issues a request for information about a word or wordform which is not listed in the online lexicon, the program attempts to analyze it as a compound. In Dutch, the constituents of nominal, verbal and adjectival compounds are written as single words — *wegverkeer (road traffic), autorijden (car driving), druipnat (soaking wet)* — and one cannot expect all compounds to figure in the lexicon. If this attempt fails, the string is considered a spelling error. The spelling checker which is subsequently activated, then proposes a small set of words as correction alternatives. The method used here, called triphone analysis (see Section 2.3 below) attains a high level of accuracy. Of course, the student always has the option of adding a word to his personal dictionary. In that case, the program initiates a dialog with the student in order to secure all information (word category, stress, irregularity, etc.) needed to compute the complete inflection paradigm. This is then stored in the lexicon and made available to future consultations of the help facility.

3. Error-diagnostic techniques

Although the three software packages were targeted on a hardware platform with limited computational capabilities, we have been able to implement three useful diagnostic tools:
—a search technique which relates manifest inflection errors to points of departure from the formal inflection algorithm
—a syntactic parser for detecting syntactic agreement errors, and
—an accurate spelling checker which can deal with sound-based errors.
The first program has been written in Pascal, the two latter ones in C language.

3.1 Inflection errors

The diagostic module in our training program for verb conjugation is based on built-in morphological and orthographical knowledge. As soon as a student types an incorrect verb form as the final response to an exercise, the program attempts to pinpoint the step in the algorithm where the student was led astray, that is, the point where the student's solution path began to deviate from the correct solution path computed by the system itself. To this purpose the program computes all possible incorrect solution paths which yield the character string typed in by the student. (In the ITS literature this approach is called 'bug generation'; cf. [2, 11] this volume.)

A solution path is incorrect if one or more of the following actions are taken:
(1) an incorrect decision at a node of the morphosyntactic decision tree,
(2) application of an illegal morphosyntactic formula (e.g. using a mixture of past and present tense affixes)
(3) performing an incorrect string modification during the orthographical stage (e.g. adding an incorrect allomorph, regularizing an irregular verb), or

(4) applying the (correct) solution path to an infinitive which is similar to but different from the one propounded in the current exercise (cf. the *wennen/wenden* example in Section 1.1).

Actions of types (2) and (3) presuppose a small list of 'malrules' which the program takes into consideration while generating erroneous verb forms. The malrules we built into the diagnostic module are based on frequent errors of native speakers of Dutch. In a second language teaching context the set should probably be extended, and partly replaced, by malrules based on prototypic errors L2 learners make. When the program embarks on an action of type (4), it chooses from a list of over 1500 Dutch verbs, including all frequent verbs.

If the program establishes several incorrect solution paths to a student response, it selects the path having the earliest point of departure from the correct path, and proposes the student to backtrack to that step of the algorithm. Due to hardware limitations (512 kB of RAM), we have made no attempt to model the student in a formal way: The program is unable to take the student's error history into account, nor is it capable of selecting or generating exercises which could differentiate between alternative diagnoses. Nevertheless, the student is often referred back to a reasonable point in the algorithm. And despite the fact that the search space of incorrect solution paths runs into the thousands, the time taken to compute the diagnosis is well below a second except for very unusual and deviant errors.

3.2 Syntactic agreement errors

The grammar curriculum software includes an LR(1) parser — a member of the family of shift-reduce parsers — similar in design to the one developed by M. Tomita [12][2]. Agreement errors are recognized and diagnosed by two special mechanisms as follows. To begin with, the non-terminal symbols of the phrase-structure grammar have been augmented with feature matrices. For example, associated with each of the symbols NP and VP there is a matrix specifying number and person features. When the automaton reduces NP and VP to S (that is, applies the rule S=>NP+VP), it checks whether the features listed in the NP and VP matrices have compatible values. Technically, this involves the execution of a *unification* algorithm to the NP and VP matrices. Unification succeeds not only in case all features match, but also when there are feature violations ('constraint relaxation'). In the latter case, a diagnostic flag is attached to the resulting subtree. For instance, suppose a student assembles the erroneous sentence *Peter word bedankt ('Peter am thanked'*; the correct spelling is *wordt* (Eng. *is)*— *word* and *wordt* are homophonous). The NP and VP matrices now specify incompatible values for the person feature (NP 3rd, VP 1st). Unification nevertheless 'succeeds' and the S node is marked as violating subject-verb agreement. This mark then triggers a feedback message to the student.

The second mechanism for dealing with agreement errors utilizes 'malrules' included in the grammar. For instance, there are rules saying that strictly transitive verbs may nevertheless occur without a direct object, and strictly intransitive verbs *with* a direct object . However, whenever the parser applies any of these rules, an error flag is put up which subsequently elicits a feedback message. The parser's diagnostic component employs a system of numerical error weights in order to obtain a quality ranking within the set of alternative parse trees computed in response to an input sentence. Any feedback messages, including automatic corrections, are based on the highest-quality tree (least errors).

[2]The parser uses a few hundred augmented phrase-structure rules which are converted to a nondeterministic pushdown automaton. Parse trees constructed by this automaton (usually several trees for one sentence) are efficiently represented in memory as 'packed forests'. For more technical details, see [12].

3.3 Spelling errors

The student wordprocessor employs triphone analysis [13] as the tool for correcting misspelled words. This technique handles both 'look-alikes' (typographical errors, e.g. letter reversals, insertions, omissions) and 'sound-alikes' (incorrect but homophonous spellings). Triphone analysis requires phonological representations of input words as well as of all words in the online lexicon. These are provided by a grapheme-to-phoneme converter [3]. Input strings typed in by students are matched against lexical entries in terms of the corresponding phoneme strings rather than the original character strings. The matching algorithm starts out by dividing the phoneme string representing the input word into overlapping segments of three phonemes. An English example is the input non-word *lites* which, after grapheme-to-phoneme conversion, is divided into the following triphones:

(l) # l aI, (2) l aI t, (3) aI t s, (4) t s #.

Each of these triphones is then looked up in a file listing all words whose phonological representation contains that triphone. For instance, the first triphone occurs in *library, light, lights, lying;* the third one is shared by *bytes, lights, writes, heights.* Similarity between input word and a lexical entry is then determined by counting the number of triphones they have in common. For instance, *lites* shares two triphones with *rights,* but only one with *slight* and *sleight.* (I leave out a few additional similarity criteria of lesser importance, e.g. length of the original character strings and left-to-right order of shared triphones.) The result is a small set of correction alternatives (in the order of 1 to 5) for most of the words. The response time is never more than a few seconds.

4. Discussion: word level error diagnosis in guided sentence production

In passing, we have touched upon three styles of error diagnosis. The parser and the spelling checker (triphone analysis) aim at what I will call *descriptive* error diagnosis: They characterize the nature of manifest errors without trying to track down bugs in the cognitive mechanism which generated them. This style is of great practical importance because it provides the student with relevant and immediate error feedback without imposing heavy demands on hardware and software. However, its impact on the learning process is bound to be rather superficial: treatment of symptoms. In contrast, *explanatory* error diagnosis does attempt to characterize a malfunctioning cognitive mechanism. This style is exemplified by the verb conjugation teaching program.

All diagnostic decisions taken by the systems we have discussed are based on *single errors.* It need not be argued that diagnostic accuracy can greatly profit from considering *multiple errors* made by the current student or by a group of similar students. This applies in particular to descriptive diagnosis. However, multiple-error diagnosis is usually practiced in combination with explanatory error diagnosis and goes by the name 'student modeling'.

Pijls, Kempen & Janner [10] propose to use the 'extended overlay' approach [14, p.347] to student modeling in the context of grammar instruction. Basic ingredient is a 'curriculum network' which specifies didactic dependency relationships between grammatical concepts (cf. [6] on genetic graphs, and [1]). For instance, our grammar teaching program introduces and explains the notion of grammatical subject in terms of finite verb, which in turn presupposes ('is dependent on') the concept of verb tense. By making fully explicit all dependencies implied by the didactic method one arrives at a network of concepts spanning the complete grammar curriculum. The current grammatical knowledge of a student can then be characterized as that part of the curriculum network which s/he has mastered. The network will

also include 'maldependencies' which express frequently observed misconceptions concerning the properties and definitions of grammatical concepts (see [10] for some details).

My chief motivation for presenting this 2*2 typology, however, is to emphasize the existence of alternatives to student modeling which uncover much useful error-diagnostic information—maybe most of the information language teachers really need—at a fraction of the cost. Rather than spending inordinate amounts of time and money on student modeling in ITSs for isolated and small-scaled L1 or L2 skills (as is current practice), one can more profitably develop simple diagnostic modules which, through the implementation of special malrules, are attuned to typical errors of well-defined L1 or L2 learner groups.

Such facilities could be installed in language training modules which offer *guided sentence production* exercises, that is, exercises where sentence content, lexical material and/or syntactic form are at least partially under program control. Examples are translation, transformation, sentence combining, sentence completion, cloze test, sentence construction by selecting words or word groups from a menu, picture description, etc. Teachers can arrange such exercises in a way granting students some freedom of shaping their written texts without setting the program diagnostic tasks which, given the limited computational resources available to many schools, would be too difficult.

Acknowledgement: The programs described in this paper have been implemented by the following members of NICI's Language Technology Project: Edwin Bos, Elena Jongen-Janner, Theo Vosse, and Ron van Wieringen. I thank them for their invaluable contribution.

References

1. Brecht, B., G. McCalla, J. Greer, & Jones, M.: Planning the content of instruction. In: Proceedings of the 4th International Conference on AI and Education, Amsterdam 1989.
2. Chanier, T., Pengelly, M, Twiddle, M. & Self, J.: Conceptual Modelling in Error Analysis in Computer-Assisted Language Learning. This volume.
3. Daelemans, W.: Studies in Language Technology. An object-oriented computer model of morphophonological aspects of Dutch. Dissertation, University of Louvain 1987.
4. De Smedt, K.: Incremental sentence generation. Dissertation, University of Nijmegen 1990.
5. De Smedt, K. & Kempen, G.: Segment Grammar: a formalism for incremental sentence generation. In: Natural language generation in Artificial Intelligence and Computational Linguistics. (C. Paris, W. Swartout & W. Mann, Eds.). Boston/Dordrecht/London: Kluwer Academic Publishers 1991.
6. Goldstein, I.: The genetic graph: a representation for the evolution of procedural knowledge. International Journal of Man-Machine Studies, 11, 51-77 (1979).
7. Kempen, G. & Hoenkamp, E.: An Incremental Procedural Grammar for sentence formulation. Cognitive Science, 12, 201-258 (1987).
8. Miller, G. & Fellbaum, C: Wordnet and the Organization of Lexical Memory. This volume.
9. Pijls, F., Daelemans, W. & Kempen, G.: Artificial Intelligence tools for grammar and spelling instruction. Instructional Science, 16, 319-336 (1987).
10. Pijls, F., Kempen, G. & Janner, E.: Intelligent modules for Dutch grammar instruction. In: Research on computer-based instruction. (J. Pieters, P. Simons & L. de Leeuw, Eds.). Amsterdam/Lisse: Swets & Zeitlinger 1990
11. Tasso, C, Fum, D. & Giangrandi, P.: The Use of Explanation-Based Learning for Modeling Student Behavior in Foreign Language Tutoring. This volume
12. Tomita, M.: Efficient parsing for natural language: a fast algorithm for practical systems. Dordrecht: Kluwer Academic Publishers 1986.
13. Van Berkel, B. & De Smedt, K.: Triphone analysis: a combined method for the correction of orthographical and typographical errors. In: Proceedings of the second conference on applied natural language processing, Austin TX. American Association for Computational Linguistics 1988
14. Wenger, E. Artificial Intelligence and tutoring systems. Los Altos, CA: Morgan Kaufmann 1987