Instructional Science 16: 319–336 (1987)

Artificial intelligence tools for grammar and spelling instruction

FIENY PIJLS, WALTER DAELEMANS & GERARD KEMPEN
Department of Experimental Psychology, University of Nijmegen, Postbox 9104, 6500 HE Nijmegen, The Netherlands
AI-Lab, Free University of Brussels, Pleinlaan 2, 1050 Brussels, Belgium

Abstract. In The Netherlands, grammar teaching is an especially important subject in the curriculum of children aged 10-15 for several reasons. However, in spite of all attention and time invested, the results are poor. This article describes the problems and our attempt to overcome them by developing an intelligent computational instructional environment consisting of: a linguistic expert system, containing a module representing grammar and spelling rules and a number of modules to manipulate these rules; a didactic module; and a student interface with special facilities for grammar and spelling. Three prototypes of the functionality are discussed: BOUWSTEEN and COGO, which are programs for constructing and analyzing Dutch sentences; and TDTDT, a program for the conjugation of Dutch verbs.

1 Grammar and spelling instruction: problems

In The Netherlands and in Flanders, grammar teaching is an especially important subject in the curriculum of children aged 10-15. Three reasons are the following:

1 Grammatical knowledge is generally considered necessary to improve the quality of writing skills.
2 The orthography of Dutch depends to a considerable extent on grammatical relationships within the sentence. In many cases the correct spelling of a word cannot be derived from memorized word spellings or word-based phoneme–to–grapheme rules. A typical example is provided by the Dutch homophonous passive auxiliaries *word* and *wordt.* Choosing between them presupposes the explicit application of subject-verb agreement rules.
3 In secondary schools at least three foreign languages are taught (French, English and German). A fair amount of grammatical knowledge is needed here, *e.g.* for determining case endings of German adjectives and nouns.

For these reasons, much attention and time is invested in grammar and spelling teaching. Nevertheless, the results are poor. In our opinion this is caused by, among other things,

- the fact that grammar is presented as a set of rules for *analysing* rather than *generating* sentences. This causes a low level of transfer from grammar instruction to writing skills. Many students are unable to apply the grammar and spelling rules they have been taught, when composing an essay;

- uninformative feedback. After making errors, students very often just write down corrections given by the teacher, and do not attempt to find reasoned solutions;
- dull and uninspiring exercises, which can be executed without much thinking on applicable rules;
- lack of notational uniformity of current school grammars (*e.g.* several teachers in one school using different or even conflicting notations);
- insufficient explicitness concerning the relations between grammatical *functions* (*e.g.* subject, direct object, predicate) and grammatical *categories* (phrases, constituents, parts of speech).

For a a detailed overview of the problems of grammar and spelling teaching in Dutch, see Van Dort-Slijper (1984) and Assink (1983).

2 Grammar and spelling instruction: information technological tools

Several attempts have been made to solve these shortcomings of grammar and spelling instruction by means of traditional Computer Aided Instruction. Although some problems can be solved in this way (immediate, consistent, and informative feedback is made possible, for instance), CAI cannot bring a complete solution. (Kempen, Schotel and Pijls (1985) evaluate the literature on CAI programs for grammar and spelling teaching in Dutch and other languages.) The major drawback of these programs is their lack of grammatical knowledge. This makes it impossible to produce a sufficiently detailed diagnosis of the learner's problems, and to come up with explanatory feedback upon errors made.

In a four-year research project which was started in September 1985, we are attempting to overcome some of these limitations by developing an intelligent computational instruction environment for grammar and spelling. It consists of three main components:

1. A linguistic expert system, which is knowledgeable about the grammatical and orthographic subject matters, that are taught to the students. Subcomponents are, among other things, a sentence generator and a parser using a notation for syntactic structures which comes close to notations that are in use in popular school grammars but is sufficiently formalized. To this purpose we devised a simplified version of Kempen and Hoenkamp's (1987) Incremental Procedural Grammar (IPG). This grammar is accompanied by a (two-dimensional) tree notation for syntactic structures which is much more transparent than the one-dimensional notations currently in use in Dutch school grammars.
2. A didactic module including a subcomponent for diagnosing the learner's knowledge, a bug catalogue (a list of frequent mistakes to guide the diagnosis

procedure), a tutoring module (knowledge about what knowledge to impart to a particular learner and how to do it), and an exercise generator.

3. A student interface. An important component of this interface is TREE DOCTOR (written by Desain, 1985), a powerful graphical tree editor. It can be used not only for displaying IPG parse trees, but also for manipulating them in various ways. TREE DOCTOR allows the student to modify syntactic trees (*e.g.* deleting and adding branches and nodes, attaching branches to other nodes, changing left-to-right order of nodes, etc.) according to the "direct manipulation" style of user interfaces advocated by Shneiderman (1982). An example of modifying an IPG parse tree is given in Figure 1. It shows a student moving the attachment point from one node to another, thus correcting an unintended parse tree of an ambiguous sentence. TREE DOCTOR also offers a facility for "closing" trees, i.e. for hiding the details of a subtree into a triangle. By asking TREE DOCTOR to display certain parts of a tree in a closed or open manner, the teacher can "zoom in" onto those aspects of the tree that are most relevant for a certain student or group of students.

Presently, the system is far from being fully implemented. However, we have finished three prototypes embodying part of the functionality that we are aiming at. One is BOUWSTEEN (building block), a program allowing students to compose sentences out of parts of speech. BOUWSTEEN runs on an IBM-AT computer. A second program, COGO (Schotel and Pijls, 1986), allows learners to practice sentence analysis on sentences they have composed themselves. A third prototype is TDTDT (TDTDT Diagnoses Trouble with DT; Daelemans, 1987), a small teaching environment for the conjugation of Dutch verbs. Both COGO and TDTDT run on a Symbolics Lisp Machine.

3 BOUWSTEEN

BOUWSTEEN (work by Huls) is meant to be the first step in the grammar curriculum. It aims at making students aware of the constituent structure of sentences; that is, of the fact that sentences can be put together by combining fixed word groups in certain left-to-right sequences. Students thus can develop a notion of word order constraints. In BOUWSTEEN this knowledge is taught in a completely different way from the traditional one. In traditional grammar instruction, students usually learn to split up given sentences into parts, *e.g.* by putting bars between constituents. In BOUWSTEEN they have to build sentences out of parts of speech belonging to all the categories. The word groups are stored in a database. A random selection from them is displayed in a menu on the screen. The student may construct any sentence he or she likes by selecting (pressing cursor keys) appropriate word groups in a correct order. He is allowed to use as many constituents as he likes. The constituents chosen are displayed at the top window

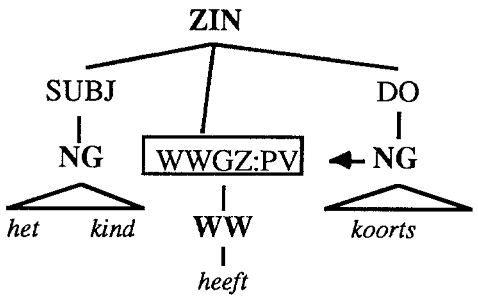


Fig. 1a. The student clicks with the mouse on the finite verb. He wishes to move the verb into front position in order to make an interrogative sentence.

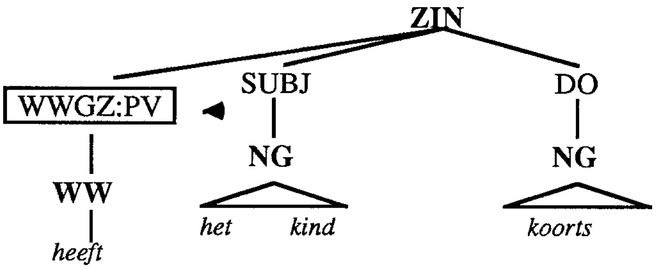


Fig. 1b. The student moves the finite verb by dragging it across the screen (direct manipulation)

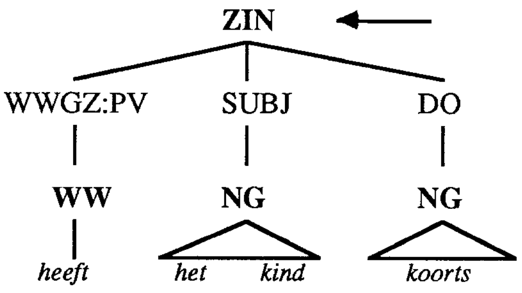


Fig. 1c. The student leaves the verb approximately at the desired position.

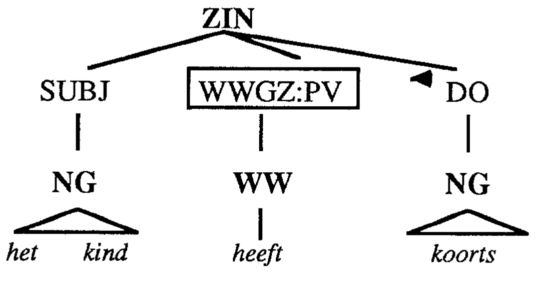
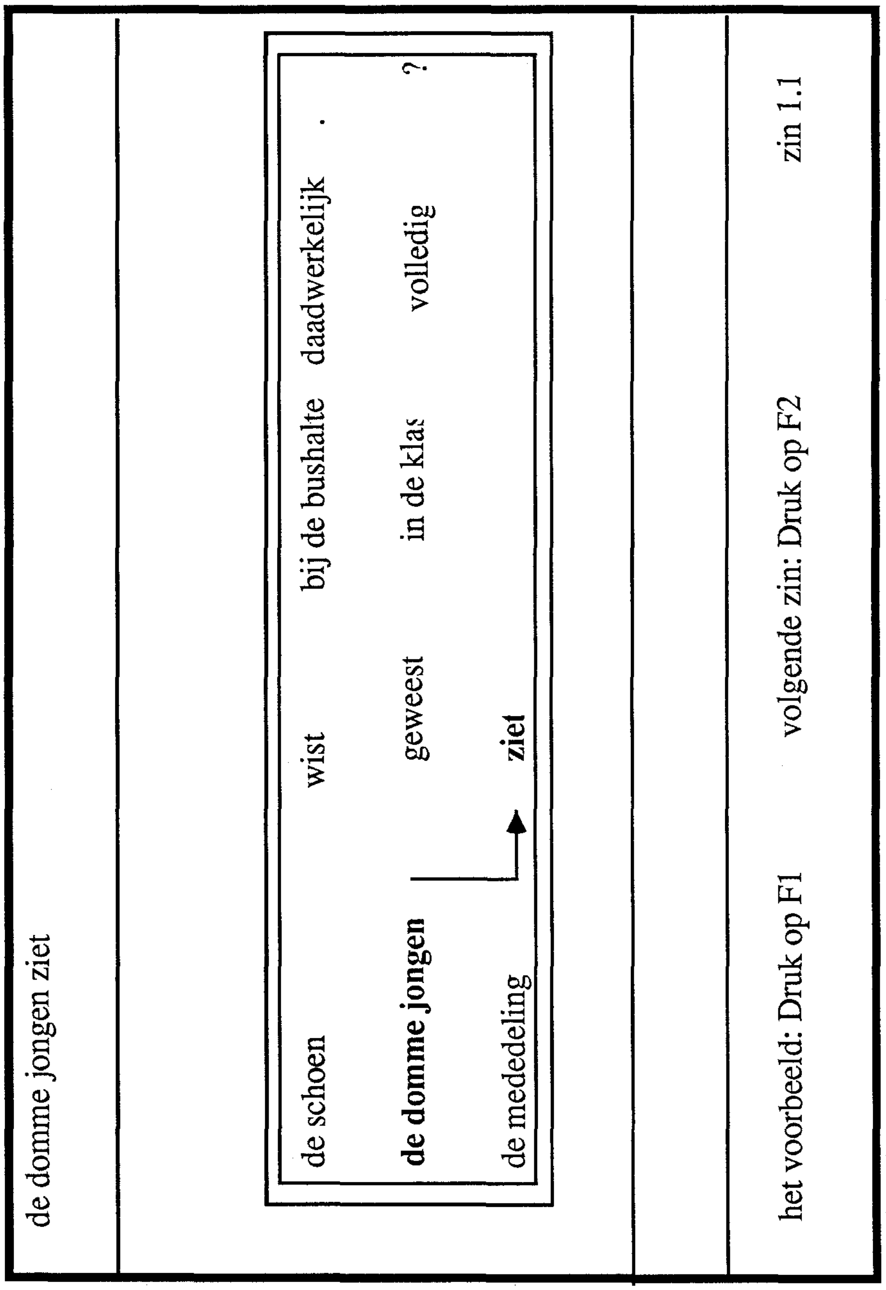


Fig. 1d. TREE DOCTOR computes the final shape of the tree, with all nodes correctly aligned.



de domme jongen ziet

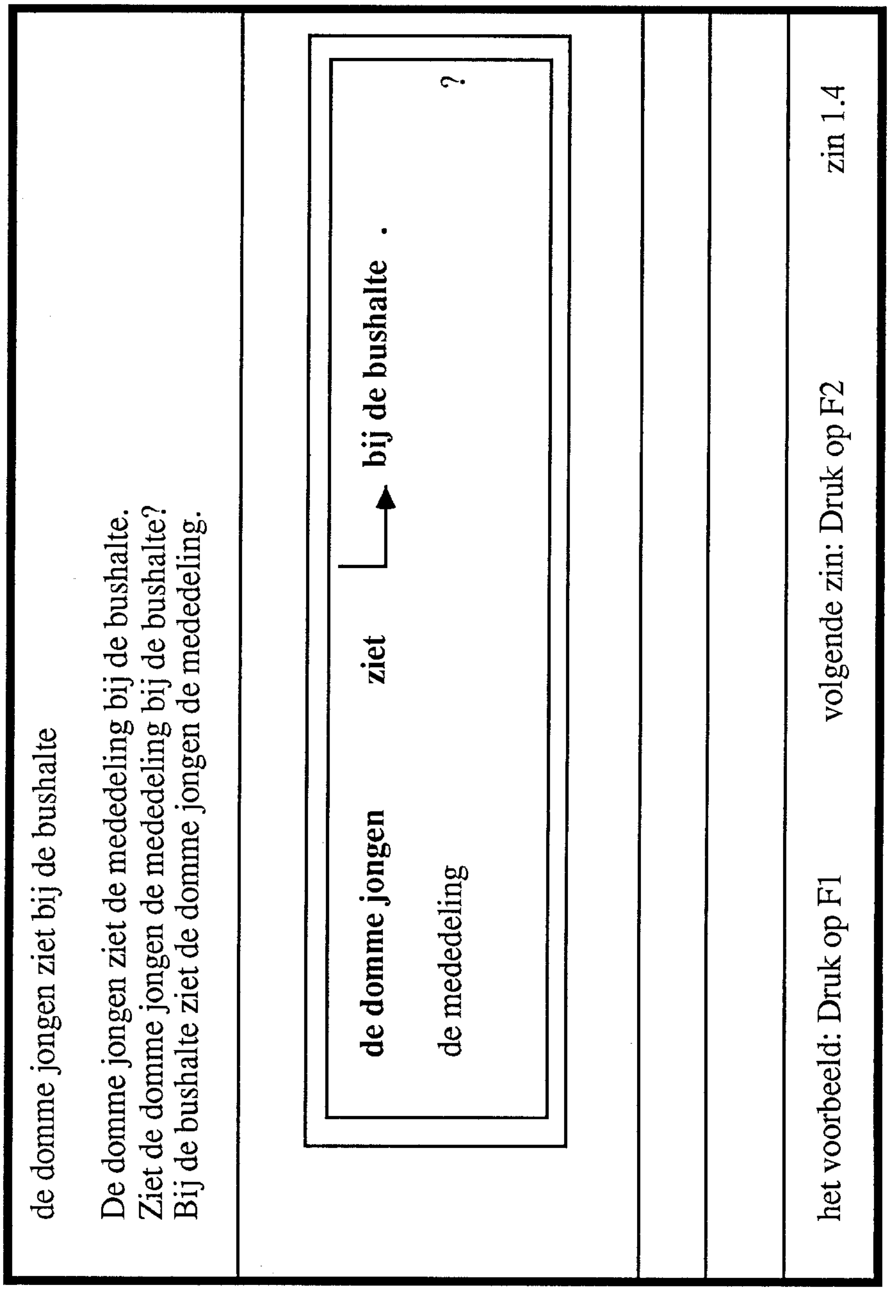
de schoen wist bij de bushalte daadwerkelijk .

de domme jongen geweest in de klas volledig ?

de mededeling **ziet**

het voorbeeld: Druk op F1 volgende zin: Druk op F2 zin 1.1

Fig. 2 BOUWSTEEN : Screen 1. The student is building a sentence by selecting word groups from the menu displayed in the middle rectangular window.


de domme jongen ziet bij de bushalte

De domme jongen ziet de mededeling bij de bushalte.
Ziet de domme jongen de mededeling bij de bushalte?
Bij de bushalte ziet de domme jongen de mededeling.

de domme jongen **ziet** → **bij de bushalte** .

de mededeling ?

het voorbeeld: Druk op F1 volgende zin: Druk op F2 zin 1.4

Fig. 3 BOUWSTEEN : Screen 2. The student puts together as many correct sentences as possible out of the word groups selected in Figure 2

of the screen (Figure 2). The sentence is analyzed by the IPG parser (written by Konst, 1986) in just a few seconds. If the sentence is grammatically correct, a second screen appears. In the menu displayed on this screen, the student only finds the constituents used in the sentence he put together (Figure 3). With these, he has to build as many additional sentences as possible. The IPG parser checks whether (1) all constituents are used in each new sentence, (2) the sentence is grammatically correct, and (3) the sentence is different from the former one(s). Whenever the student wishes to work with a new set of constituents, because he dislikes the set given (on screen 1) or he has made enough variations (on screen 2), he can ask for another menu. Whenever a student makes an ungrammatical sentence on screen 1 or screen 2, this is signalled by the parser. Then, the ungrammaticality is explained to the student at an appropriate level.

BOUWSTEEN also contains a pop-up menu, which gives the student the possibility of asking for examples and explanation. When the student stops a BOUWSTEEN session, he can get a summary of the subject matter taught. For example, it is explained that the word groups are called *zinsdelen* ("sentence parts"), and he is shown a rudimentary tree structure in which the hierarchical relationship between a *zin* (sentence) and *zinsdelen* (constituents) is visualized (Figure 4).

Further extensions of this tree structure will be worked out in future grammar instruction programs. The closed subtrees of Figure 4 will be opened, so that their internal structure will be visualized. In this way, the student is gradually familiarized with all grammatical categories and functions in their mutual dependencies. Finally, students will be able to work with complete tree structures of the kind used in the COGO program (section 4).

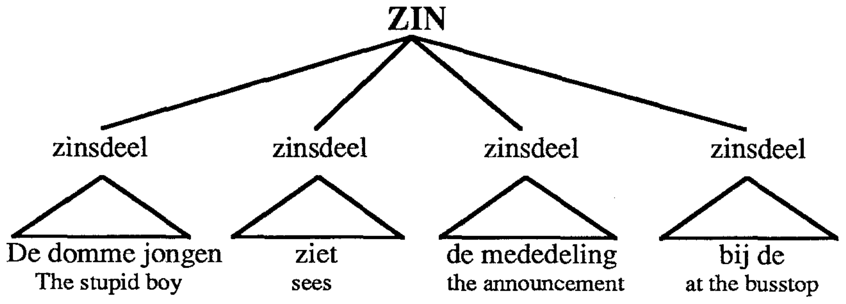


Fig. 4. BOUWSTEEN : A rudimentary tree structure visualizing the hierarchical relationship between *zin* (sentence) and *zinsdelen* (constituents)

The described method for grammar instruction is a *constructive* one: the students build and rebuild their own sentences. This contrasts with the usual *analytic* method, where students only have to label constituents of given sentences. A user test with twenty-six students aged 12 has shown that half an hour of training with a preliminary BOUWSTEEN prototype brought about results comparable to a fully developed analytically-oriented grammar instruction program, which covered roughly the same subject matter. We hypothesize that optimal results will be obtained in combinations of constructive and analytic grammar teaching strategies. An example of such a program is COGO.

4 COGO

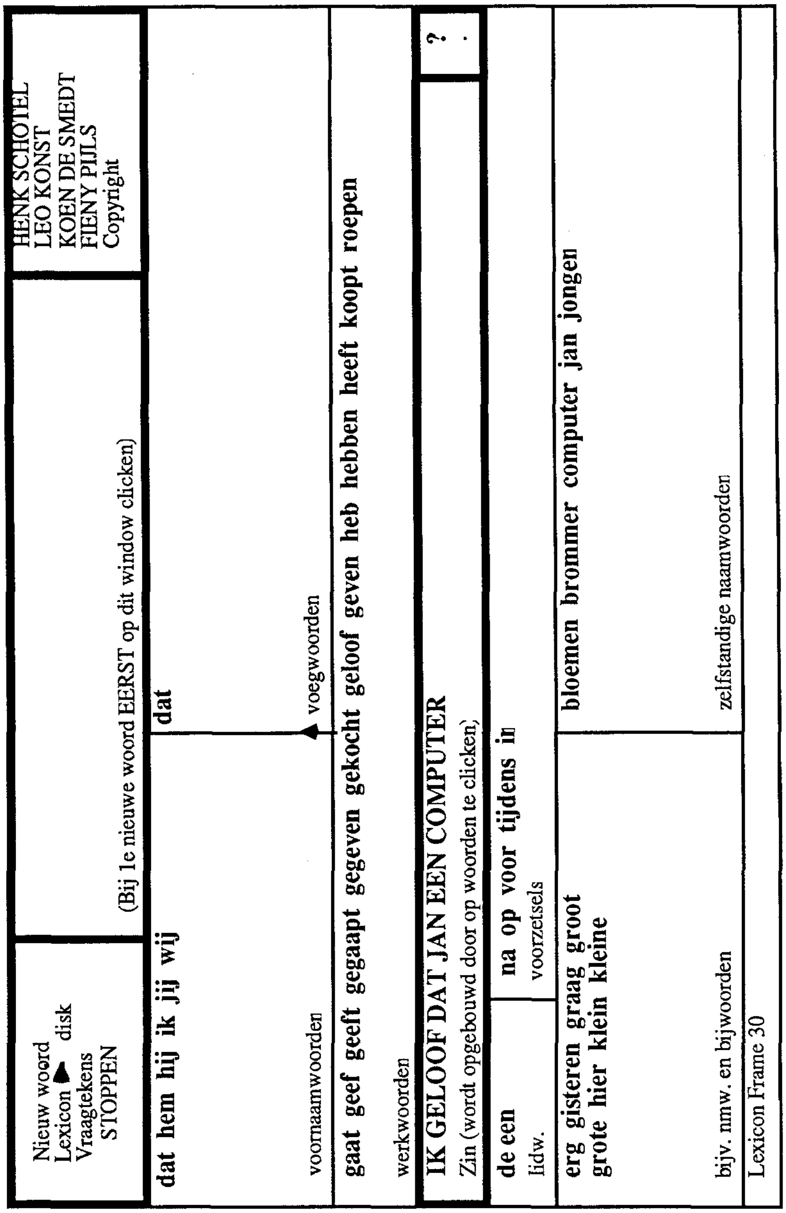
COGO, which has been developed as a stimulating training environment for syntactic structure analysis, can be used for three different didactic purposes. It enables students to

- inspect the structure of newly constructed sentences displayed in a graphical tree notation;
- present themselves with practice trials on labeling the syntactic functions, constituents and word categories in sentences which they construct out of a vocabulary of words displayed on the screen;
- explore the morpho-syntactic properties of words and word classes.

A session in COGO runs as follows. At the beginning, the student finds the screen divided into non-overlapping windows (Figure 5).

Every window contains a number of words belonging to one word class. Homographs belonging to different word classes may be inserted in different windows. In the wide horizontal rectangle in the middle of the screen, the student can "write" a new sentence. To do this he moves the mouse to the words he needs for the sentence he has in mind. When clicking on the left button of the mouse, the selected word appears in the rectangle to the right of any words selected earlier. If he needs a word that is not displayed in any window, he can type this new word into the empty rectangle at the top the screen. Then the system requests him to define the syntactic properties of the word (which are needed by the parser) via a sequence of menus. After this, the new word appears in the appropriate window, and the student can use it in his sentence. Clicking on the period or the question mark signals the IPG parser (the same one as used in BOUWSTEEN) to start analysing the sentence. After just a few seconds the syntactic tree - in IPG terms, with explicit alternation of categories and functions - becomes visible on the screen.

The student can initiate a grammatical labeling exercise by asking COGO to hide certain node labels, *i.e.* to replace them by question marks in the displayed



Nieuw woord
Lexicon ▲ disk
Vraagtekens
STOPPEN

(Bij 1e nieuwe woord EERST op dit window clicken)

HENK SCHOTEL
LEO KONST
KOEN DE SMEDT
FIENY PIJLS
Copyright

dat hem hij ik jij wij
voornaamwoorden

dat
voegwoorden

gaat geef geeft gegaapt gegeven gekocht geloof geven heb hebben heeft koopt roepen
werkwoorden

IK GELOOF DAT JAN EEN COMPUTER
Zin (wordt opgebouwd door op woorden te clicken)

? .

de een
lidw.

na op voor tijdens in
voorzetsels

erg gisteren graag groot grote hier klein kleine
bijv. nmw. en bijwoorden

bloemen brommer computer jan jongen
zelfstandige naamwoorden

Lexicon Frame 30

Fig. 5. COGO: The lexicon frame

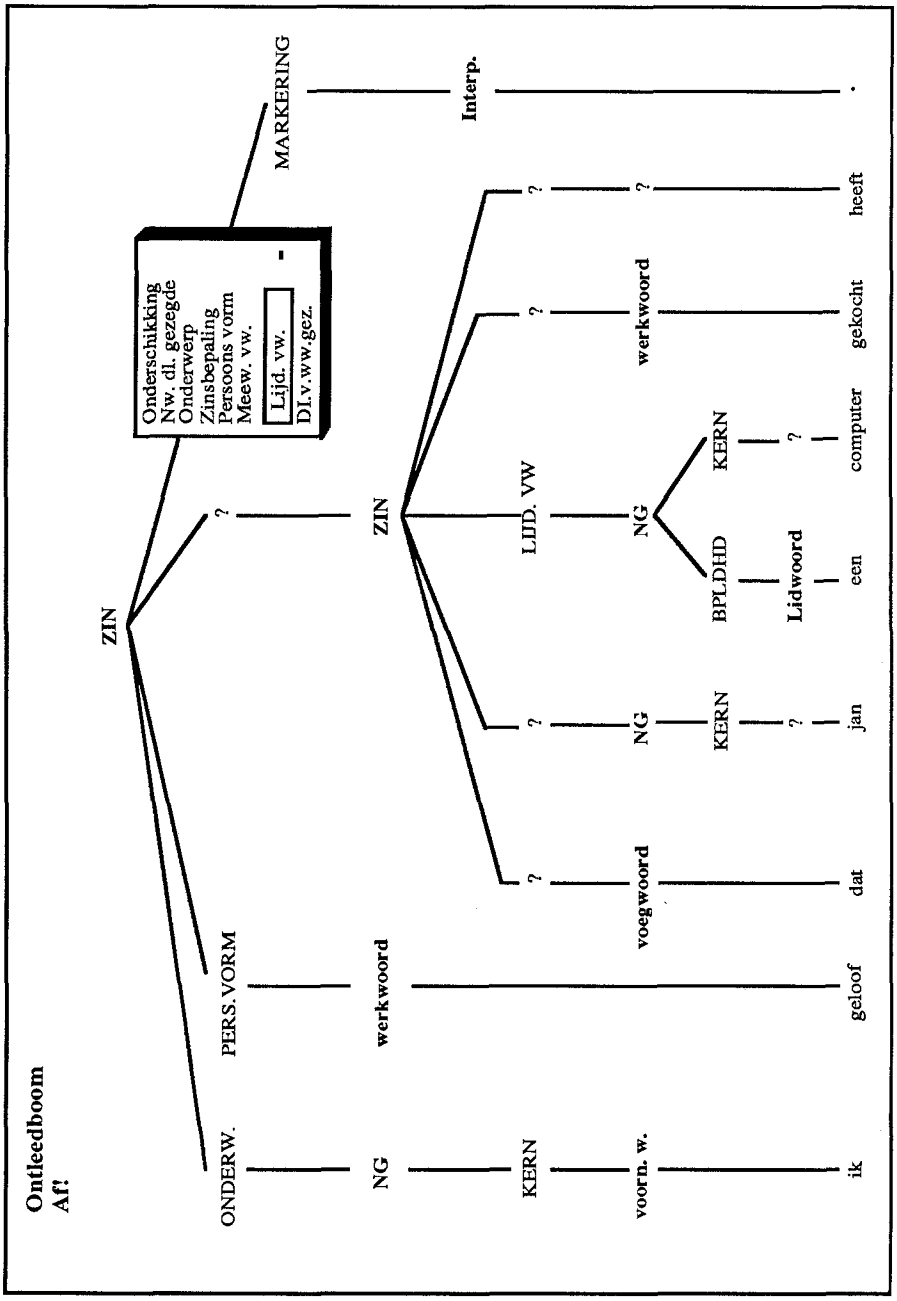


Fig. 6. Tree structure with question marks (some already replaced) and a menu for syntactical functions

tree. The student decides which type(s) of labels he wants to hide, *e.g.* all labels, just word class labels, or syntactic function labels at the clause or word group levels. In Figure 6 the student has asked COGO to hide syntactic functions and word classes. After clicking on a question mark, a menu pops up displaying a small list of labels. The labels listed depend on the place of the question mark in the tree.

There are four different menus: syntactical functions at the clause level, word groups (phrases), functions at the word group level, and word classes. The student can select a label by moving the mouse. Short definitions of the selected label appear at the bottom of the screen. When the student has indicated his choice by pushing the appropriate mouse button, COGO compares the selected label with the label computed for that node by the parser. In case of a correct choice, the question mark is replaced by the selected label. If not, an error message appears on the screen. In its present form, COGO cannot yet give any further feedback. After the student has thus removed all question marks, or if he likes to stop with a particular sentence, he can click on the word "AF!" (=done!). The tree then disappears and the lexicon windows are displayed again. In case of ambiguity, COGO displays more than one tree, and the student can select the tree that he considers appropriate. We have used COGO in an informal evaluation study. We observed that students aged 14-15 found the graphical IPG notation very easy to use, and that they were motivated by the opportunity of devising their own sentences.

5 TDTDT

A third domain of application of our intelligent teaching environment is a problematic aspect of Dutch spelling: the conjugation of verbs. We designed a complete representation of the morpho-syntactic rules which make up the domain of knowledge, and developed modules enabling students to practice their domain knowledge, for automatically diagnosing the errors students make, and an interactive user interface.

Dutch spelling is guided by two main principles: the *phonological principle* (write a word as it is pronounced in standard Dutch) and the *morphological principle* (spell related forms the same way). Unfortunately for learners of Dutch, these principles are partially conflicting. Especially in the case of some verbal inflectional endings, a lot of confusion arises (Table 1). Generally, errors involve an improper application of the phonological principle instead of the morphological principle in the spelling of verbal inflectional endings. *E.g.* learners are often at a loss whether a conjugational ending sounding like /t/ is written <dt>, <d> or <t>. The correct solution involves the application of a number of syntactic, morphological, phonological and spelling rules.

	Phonological Transcription	Lexical Representation	Spelling	Gloss	Principle
1	/bleif/	blijv	blijf	I stay	Phon
2	/lat/	laad	laad	I load	Morph
3	/lat/	laad+t	laadt	he loads	Morph
4	/lad@/	laad#de	laadde	I loaded	Morph
5	/g@leit/	ge#leid+d	geleid	lead	Phon+Morph

Table. 1. Phonological transcription, lexical representation, spelling, gloss and governing principle in a few conjugations

The domain expert of TDTDT contains the necessary linguistic knowledge, implemented as a system of concepts in the KRS knowledge representation system (Steels, 1986), to conjugate any Dutch verb. To these linguistic concepts, explanatory information was attached, and a rule-history was added to concepts representing computed verb forms. A rule-history lists the different decisions taken and rules applied to produce a particular word form, and is used in diagnosis.

The domain knowledge is presented according to the *algorithmic-rule method* (Assink, 1983), a recent development in spelling teaching. In this method, an easily manipulatable *algorithm* is devised, pictured on a card, which allows quick and easy decisions on the spelling of verb forms. First, the learner is introduced to a number of concepts playing a role in the solution of the problem and featuring on the card (tense, finiteness, person, number *etc.*). In the following stage, these concepts are related to spelling rules by means of the algorithmic decision scheme on the card. Exercises are made with the help of the card until the algorithm is internalised and the card becomes superfluous.

We have automated the second part of this teaching method (practicing the algorithmic decision scheme). We assume that learners working with our system have received classroom teaching about the different linguistic concepts used in the algorithm. For each specific conjugation problem, a decision tree is pictured on the screen. The solution of a specific conjugation problem is represented as a walk through the decision tree (Figure 7).

Each node represents a question (*e.g.* 'Is it an action in the past?') to which the learner answers by clicking on a menu item. If a wrong choice is made this is explained to the learner. Teaching material (exercises) can be generated by both the system and the learner. In the former case, either by using the on-line lexical database to select random verbs or by selecting verbs from a "difficult cases" list

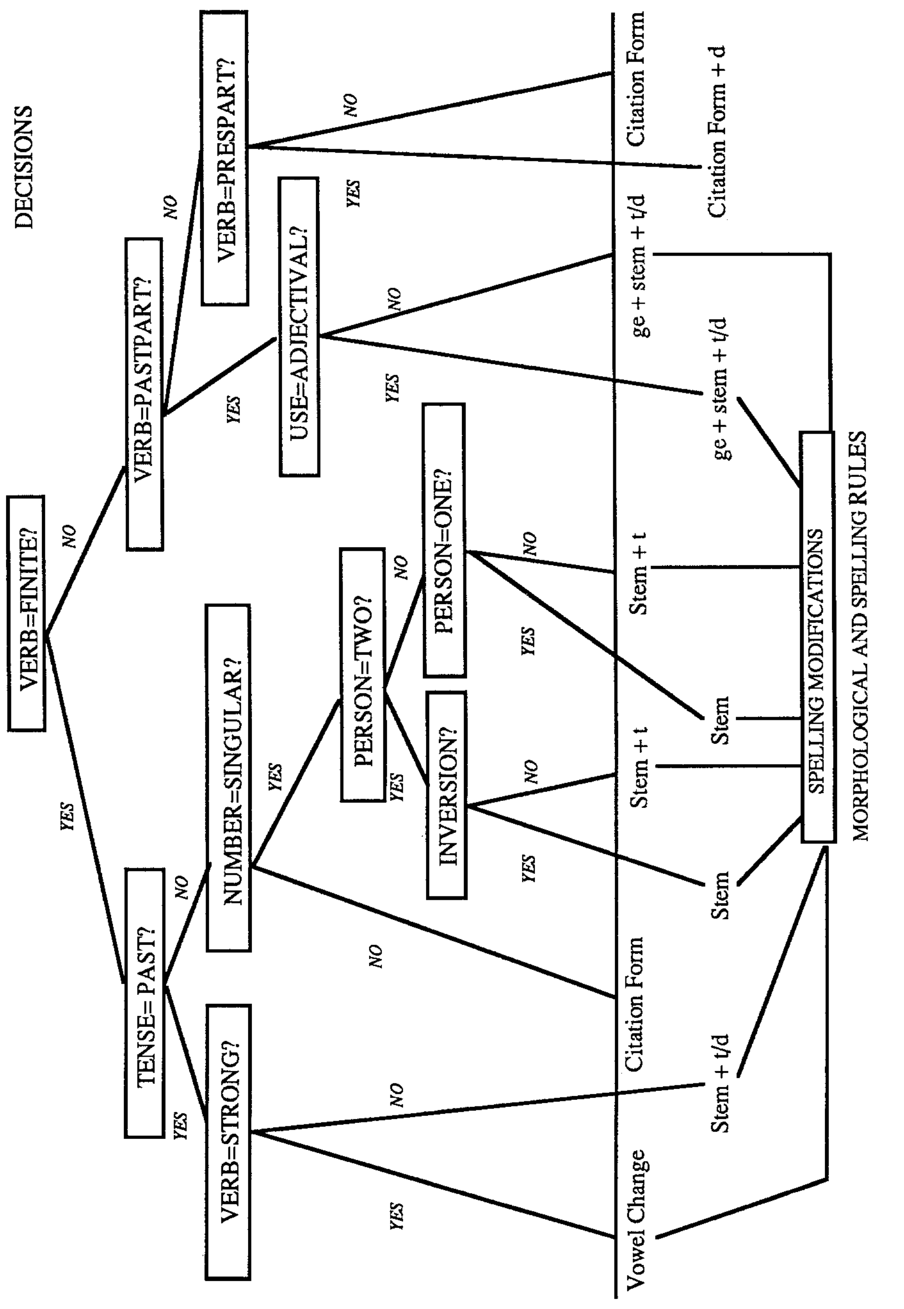


Fig. 7. Decision tree for the conjugation of Dutch verbs

explicitly programmed by the teacher. In the latter case, the learner has a free choice of the verb he wants to exercise. The infinitive (dictionary citation form) of the verb is entered, and the system computes all its conjugated forms. Next, a heuristic (based on a comparison of the phonological and the spelling representations of the verb form) is used to select those verb forms which are likely to cause mistakes. These forms are "computed" by the learner with the help of the decision tree as described above. Corrective feedback is provided whenever a wrong decision is taken.

After some time, the learner should be able to solve conjugation problems without the help of the decision tree. At this point, exercises are presented without explicit algorithmic support, and the input by the learner is compared to the forms computed by the system itself. Again, exercises may be provided either by the system or by the learner. Whenever the learner makes a mistake (*i.e.* the form input by the learner and the form computed by the system do not coincide), several things may happen. As a first possibility, the particular exercise may be repeated, this time with the decision tree. This should remind the learner of the decisions to be taken in deriving a correct answer, and may provide additional explanation if at some node in the decision tree an incorrect course is taken. However, a more challenging option is to activate at this point a diagnosis system which looks for the "bug" responsible for the mistake in the learner's internalised knowledge. Detailed diagnosis of problems is essential to be able to provide relevant additional explanation and practising material (Breuker and Cerri, 1982). We have developed an experimental version of such a diagnostic procedure which we will describe shortly.

A correct solution to a spelling problem can be represented as a sequence of correct decisions forming the conditions of a number of morphological and spelling rules. *E.g.* a correct computation of the second person singular of *redden* (to save) involves the following decisions and rules: finite=yes, past=no, singular=yes, first-person=no, second-person=yes, inversion=no, stem-computation—rule (results in the form *redd*), morphological structure-building rule for second person singular (results in *redd+t*), consonant-degemination spelling rule (results in *red+t*), result is *redt*. This sequence fills the rule-history slot of the *redt* verb form after computation of this form by the domain expert.

Rules can be uniformly represented as consisting of a number of conditions and a number of actions, performed if the conditions apply in a particular context. The source of mistakes can now exhaustively be characterised as follows:

- At the macro-level: the omission or insertion of rules or the muddling up of the correct sequence of the rules.
- At the micro-level: the omission of conditions or actions, the insertion of conditions or actions or the muddling up of the correct application order of conditions or actions. In our case, conditions are either decisions in the decision tree or conditions based on the form of the verb to be conjugated.

It will be clear from this that if we want to check all possible sources of an error, a combinatorial explosion of alternative possibilities results, even for relatively simple forms. Furthermore, the insertion of extraneous rules, conditions or actions is open-ended, and therefore unpredictable. This proves that an approach involving *exhaustive diagnosis* (like BUGGY for arithmetic; Brown and Burton, 1978) cannot be transported to more complex domains.

A solution to this problem commonly adopted is the compilation of a bug catalogue (*e.g.* Anderson and Reiser, 1985), based on an empirical study of the errors learners make, and on the relation of the results of this study to the different rules used by the domain expert. By means of the information thus acquired, it is possible to construct heuristic rules which guide the diagnosis process. However, compiling such a list is a time-consuming activity, and we believe that it can be automated to a large extent through progressive refinement of heuristic diagnosis rules acquired through interaction with learners.

Heuristics always treat a number of potentially relevant aspects of the problem (in this case diagnosis) as irrelevant. This can be justified if the number of things that actually go wrong in practice is considerably smaller than the number of things that can go wrong in principle. We believe this to be the case in the diagnosis of mistakes by learners in general and in the conjugation of verbs in particular.

We start from the rule history computed by the system. As a first hypothesis, the system assumes that the mistake is due to a wrong decision somewhere. All decisions in the list are negated in turn, and the effect of this on the computation of the verb form is considered. Those decisions, which – when negated – result in the same mistake as was made by the learner, may possibly have caused the mistake. Note that the computation of the effect of the negation of a decision often involves the postulating of other decisions. *E.g.* negating past=no generates some new decisions such as weak=yes (or no). The system assumes the correct decisions to be taken. If several possible causes remain, the system tries to choose one by asking additional questions (*E.g.* 'Is it a weak verb?') or by generating additional exercises. These additional questions and exercises are by no means unnatural to the learner because they take the same form as the "normal" questions and exercises while interacting with the system.

The number of possibilities can then be constrained by computing the intersection (if the answer was again wrong) or the difference (if the answer was correct) of the relevant lists of decisions. If no hypotheses remain, *i.e.* all decisions taken by the learner are considered correct, the bug must be due to a mis-application of one of the rules. Roughly the same method can then be used to identify the rule in question. From such a diagnostic interaction, a heuristic rule is derived. *I.e.* the association of a symptom (the wrong answer) with a diagnosis (a wrong decision or wrongly applied rule). Symptoms should be expressed at a suitable level of abstraction: it would not be very useful to have a specific verb as symptom. Rather the symptom is expressed in terms of the category of the verb (regular, semi-irregular or irregular classes) and the ending of the stem. In additional interactions with the same learner, already existing heuristic rules are gradually refined.

Our approach to rule-refinement is based on the approach to learning in second-generation expert systems (Steels, 1985; Steels and Van de Velde, 1985). Second-generation expert systems consist of a heuristic rule system (much the same as traditional expert systems) called the surface model, and an additional deep model which provides an understanding of the complete search space over which the heuristics operate. A second-generation expert system can fall back on deep reasoning when its heuristic rules fail. The most important advantage of this architecture lies in the fact that it allows automatic learning of new heuristic rules. Rule learning happens by refinement (non-monotonically): a new heuristic rule is abstracted (in our case from an interaction with the learner), and is integrated into the already existing rule-base. This integration does not make the previous rules superfluous, but may restrict their application. Two options are open in the use of heuristic rules: either a new rule base is constructed for each learner, or the same is used for all learners. In the latter case, the identity of the learner may be added to the list of symptoms.

An important facet in the development of ICAI systems is the user interface. The program should be easy to work with for children, and attractive enough to motivate the learner to work with it. In TDTDT, the decision tree is built up in interaction with the learner. At each node, a question is asked, and the answer (mostly yes or no) is given by clicking with the mouse on a menu of options. If the answer is correct, a branch and new node are drawn, and a new question is asked. If the answer is wrong, the learner is told so, and some explanation is given.

In an exercise (the computation of the correct form of a verb), the infinitive of a verb is presented in a syntactic context. The generating of these contexts is still a problem. Only limited information about the case frame associated with the verb is available from the lexical database, and no semantic information at all. Contexts are therefore very simple and canned phrases designed to be applicable to (almost) all verbs. The problem can be partly overcome by providing a more detailed and natural context and letting the learner choose an applicable verb himself (this limits his free choice, but not to a point where only one or a few possibilities are left open). An alternative solution would be to explicitly ask for a specific form (*e.g.* 'What is the inflected past participle of the verb *werken* (to work)?'). In the latter case, a lot of the decisions that must be taken in the real-life computation of verb forms are given in the question, which diminishes the functionality of the tutoring environment considerably.

6 Conclusion

Intelligent teaching environments like BOUWSTEEN, COGO and TDTDT may become important steps in the liberation of class teaching from the less interesting aspects of the subject matter and in the adaptation of the instruction to the needs and pace of individual learners. We have shown that such systems can be developed even for subject matters traditionally considered "complex". It should be emphasized that the described instructional tools rely on sophisticated hardware and software resources (with the possible exception of BOUWSTEEN, which runs on an IBM-AT with 3 MByte of internal memory), presently out of reach for most schools. However, we are confident that the evolution of the microcomputer market will make possible the implementation of similar programs on cheaper systems.

Notes

This is a slightly adapted and extended version of an article which has appeared in the Nederlands Tijdschrift voor de Psychologie en haar Grensgebieden, Volume 42. An earlier version of this paper was presented at the European Seminar on ITS, Tubingen, Germany, October 25–31, 1987.

References

Anderson, J. and Reiser, B. (1985). The LISP TUTOR. *Byte, 10,* 4, 1985.

Assink, E.M.H. (1983). Leerprocessen bij het spellen: aanzet voor de verbetering van de werkwoordsdidaktiek. Dissertation University of Utrecht.

Breuker, J. and Cerri, S. (1982). A new generation of teaching machines: intelligent and rather-intelligent computer assisted instruction discussed and exemplified. In E. van Hees and A. Dirkzwager (eds.), *Onderwijs en de nieuwe media.* Lisse: Swets and Zeitlinger.

Brown, J.S. and Burton, R.R. (1978). Diagnostic models for procedural bugs in basic mathematical skills. *Cognitive Science, 2,* 155–192.

Daelemans, W. (1987). Studies in Language Technology. An Object-Oriented Computer Model of Morphophonological Aspects of Dutch. Dissertation, University of Leuven.

Desain, P. (1985). TREE DOCTOR, a software package for graphical manipulation and animation of tree structures. In *Proceedings of the Symposium on Man-Machine Interaction,* Amsterdam, December 1985.

Huls, C. (1988). *Bouwstenen voor computerondersteund constructief grammatica-onderwijs.* Master's thesis, University of Nijmegen.

Kempen, G., Schotel, H. and Pijls, F. (1985). Taaltechnologie en Taalonderwijs. In J. Heene and Tj. Plomp (eds.), *Onderwijs en Informatietechnologie.* Den Haag: SVO.

Kempen, G., Anbeek, G., Desain, P., Konst, L. and De Smedt, K. (1987). Author Environments: fifth generation text processors. In *Esprit '86, Proceedings of the ESPRIT 1986 Conference.* Amsterdam: North-Holland.

Kempen, G. and Hoenkamp, E. (1987). An Incremental Procedural Grammar for sentence formulation. *Cognitive Science, 12,* 201–258.

Konst, L. (1986). A syntactic parser based on filtering. Paper, University of Nijmegen.

Schotel, H. and Pijls, F. (1986). Een prototype van grammatica-onderwijs op een LISP-machine. *Informatie, 28*-1, 48–50.

Shneiderman, B. (1982). The future of interactive systems and the emergence of direct manipulation. *Behavior and Information Technology, 1*, 237–256.

Steels, L. (1985). Second-generation Expert Systems. In *Future Generation Computer Systems*. Amsterdam: North-Holland Pub.

Steels, L. (1986). Tutorial on the KRS concept system. Paper, AI-Lab, Free University Brussels.

Steels, L. and Van der Velde, J. (1985). Learning in Second-Generation Expert Systems. In J. Kowalik (Ed.), *Knowledge-Based Problem Solving*. N.J.: Prentice-Hall Inc.

Van Dort-Slijper, M. K. (1984). *Grammatica in het basisonderwijs*. Leiden: Martinus Nijhoff.