Sentence Completion Tests for Training and Assessment in a Computational Linguistics Curriculum

Cerstin Mahlow, Michael Hess
Institute of Computational Linguistics, University of Zurich
Winterthurerstr. 190
CH-8057 Zurich,
Switzerland,
{mahlow, hess}@cl.unizh.ch

Abstract

This paper presents a novel type of test, halfway between multiple-choice and free-form text, used for training and assessment in several courses in a Computational Linguistics curriculum. We will describe the principles of the test, the different ways in which it can be used by learners, and the tools developed for authoring. Use of this type of test is not limited to the field of Computational Linguistics. Wherever text heavy or even picture based topics are taught use of this type of test is possible.

1 Introduction

Students of Computational Linguistics (CL) at the University of Zurich come from two different faculties, viz. the *Faculty of Arts* and the *Faculty of Economics, Business Management and Information Technology*. Thus they have a very uneven previous knowledge of linguistics and programming. The introductory lectures touch upon most aspects of CL but cannot compensate for these differences in a satisfactory way. We are trying to ease the problem by supplying students with extensive additional on-line reading material for individual study. However, until recently students had no way of testing the knowledge they acquired through self-study against the requirements of the courses. For this reason we developed web-based tools for individual training and self-assessment.

Most assessments in web-based learning courses use Multiple Choice (MC) tests. These tests are easy to create for authors and easy to use for students. Unfortunately the concept of MC imposes a very restrictive format on the tests, and they can basically test only the presence or absence of small "knowledge bites". More general and abstract types of knowledge are hard to test by means of MC.

Free-form text tests, i.e. tests allowing replies in the form of mini-essays, are, of course, far less restrictive but the costs of assessing them by hand is, in many institutional contexts, prohibitively high. Systems for reliable and consistent automatic assessment of free-form text are not yet available. Those that exist either test writing style, or test the presence or absence in an essay of (explicit) terms or of (implicit) concepts (example: IEA; (Landauer et al., 1998, p295-284)), or use a combination of surface lexical, syntactic, discourse, and content features (example: e-rater; (Burstein, 2003)). It was shown, by the system developers themselves, that the most advanced of these systems, e-rater, can be tricked rather easily into giving marks that are far too good, by using some knowledge of the techniques used by the system (Powers et al., 2001). Since knowledge of the techniques used by rating systems can hardly be kept secret for any length of time all such feature based systems are open to this kind of tricks.

This is why we developed a new type of test, called "Satzergänzungstests" (SET) – SET[1], positioned halfway between multiple-choice tests and free-form text tests. We use this type of test for training as well as for assessments, and it is part of our web-based curriculum. The development was funded by the University in view of the implementation of the Bachelor/Master/PhD based "Bologna scheme" in most European universities (see (European Ministers of Education, 1999)).

With SETs we are able to create far more demanding tasks for training and assessment than we could otherwise. The philosophy behind SETs will be presented in Section 2.

[1] "Sentence Completion Tests".

In Section 3 we will show how the individual student can use a test. In Section 4 we will show how to create tests. In section 5, finally, we will give an overview of the courses in which we use these tests for teaching Computational Linguistics (CL), and discuss in which other contexts they could be used.

2 The philosophy behind SETs

(Rütter, 1973) creates an extensive topology for assessments. He distinguishes between *open*, *semi-open* and *closed* tasks. The distinction derives from the type of answer expected from the learner: There is no certain answer the author expects (open tasks), the author expects a certain answer the learner has to create themselves (semi-open tasks), the learner has to choose the right answer(s) from given possibilities (closed tasks). *Multiple Choice* tasks (MC) belong to the closed tasks.

The topology presented by Rütter is not restricted to the easy tasks. You will also find so-called *"Erweiterungswahlaufgaben"* in the class of closed assigns. This task consists of a piece of information the tested person has to extend so as to create a coherent piece of new information. The learner can choose suitable extensions from a given list. Rütter's description includes the hint that these tasks are hard to design but present a very clear structure for the test person.

Our Sentence Completion Tests can be seen as an instance of such *Erweiterungswahlaufgaben*. The learner has to answer a complex question in near-free-form on the basis of extensive choices of possible answer components supplied by the system. There will be answer components considered indispensable, some considered correct but optional, others categorised as outright wrong, and others still rated as irrelevant to the question at hand. The required components of the answer will all have to occur in the answer but not in a fixed order.

In concrete terms this means that a learner will author an answer in the form of a complete sentence, in a piecemeal fashion and under the guidance of the system, by picking answer fragments from dynamically adapted choice menus popping up as the answer is growing. At each step the user input will be checked by the system against the answer model that contains all the expected answer parts, essential relationships between them, and possible restrictions on the order of these parts. At each step as well as at the very end the system can generate feedback for the user which will make him understand why and in which aspects his answer is correct or incorrect.

3 How to use a SET

All SETs are presented under a web interface. The student has to start a browser[2] and choose a single SET[3].

3.1 Basic elements of a SET

The student sees four coloured fields[4], each labeled with a number and a functional description. These fields are:

1. Text up to now
2. Comments/Feedback
3. List of elements to continue
4. Preview

Text up to now contains the question and the answer in its present state. *List of elements to continue* consists of possible continuations of the answer. Clicking on one of the radio buttons activates the *Preview* showing all the options that will become available once the learner has committed himself to the given continuation. That way the user is always aware of the consequences his choice might have. The listing field includes two submit buttons, one for submitting the choice, one for undoing the last choice. The element list will show the elements in different order each time the user reloads or restarts a SET.

The crucial field is the one for *Comment/Feedback*. The user does not merely get a "Right - Wrong" feedback but rather

- If the answer contains the correct components but a wrong relationship the feedback will point this out and invite the user to try again and find the correct combination.

[2] At the moment the SET web interface is tested with Netscape and InternetExplorer.

[3] Where to find all SETs will be described in section 5.2.

[4] The same colours are used in our ordinary MC-tests to give the student a familiar feeling for the assessment situation.

- If the answer consists of correct components as well as of wrong ones the feedback will say so and point out which components are wrong.
- If the answer is one of the correct ones, the feedback will approve this solution and mention the other possible correct answers.

This way for every possible combination of answer components the user gets a different optimised feedback.

The text inside the feedback field is displayed as HTML so that it is possible to include links to related SETs, back to the lecture notes or associated material. A feedback text also can include a link to a new SET, as a followup.

Sometimes it is useful to have the system generate a comment before a complete answer has been created by the learner. Once the learner has chosen a certain number of wrong answer components he will get suitable feedback before finishing. In this case the feedback is used to warn the user that he is on a completely wrong track and that he ought to undo some of the last choices, or to start again from scratch.

3.2 A sample SET

See figure 1 for a sample session with SET. The initial question is: *"Was ist ein Parser?"* (What is a parser?).

Here the user chose *"Ein Parser ist eine Prozedur"* ("A parser is a procedure") as next element in the third field. This will be the beginning of his answer. Clicking on the corresponding radio button activated the preview in the fourth field. Before submitting the choice, the user can think about the combinations his choice will allow. The preview shows 4 possibilities to continue with the description of the aim of this procedure.

If the user is satisfied with his choice, he will click the submit button *"Auswahl bestätigen"* (Confirm choice). This will result in reloading the site with the new information.

Text bisher (Text up to now) will contain the question, the beginning of the answer and the fragments added by the learner so far *"Ein Parser ist eine Prozedur"*. The feedback field will still be empty. *Auswahl der Fortsetzungen* (List of elements to continue) will show all possible continuations. *Vorschau* (Preview) will be empty until the user clicks on one of the radio buttons in the list of elements to continue.

This sequence of actions will be repeated until the user has created a complete sentence. He then gets the feedback. If he is not satisfied with one of his choices before finishing, he can undo the last choice, or simply restart the SET.

In case the user is on a probably wrong way he will get feedback before finishing the SET. See figure 2 for an example. The user created an answer start *"Ein Parser ist eine Prozedur, um die syntaktische Korrektheit einer Sprache..."* ("A parser is a prozedure to ... the syntactical correctness of a language"). The intervening feedback points to the principle of correctness concerning certain constructions of languages and prompts the user to undo the last decission(s). (*"Was wohl soll die syntaktische Korrektheit einer Sprache sein?! Nur einzelne Konstruktionen einer Sprache können korrekt oder inkorrekt sein. Einen Schritt zurück!"*

Figure 3 shows the finished SET. The user followed the hint in the intervening feedback shown in figure 2. He removed the part *"einer Sprache"* ("of a language"). The answer created by the user is *"Ein Parser ist eine Prozedur um die syntaktische Korrektheit eines Satzes zu ermitteln"* ("A parser is a procedure to detect the syntactical correctnes of a sentence"). Clearly, this answer is not correct. It describes rather an acceptor than a parser. The comment says so and then offers a correct definition with a hint to the latin origins of *Parser*.

3.3 Training mode and assessment mode

SETs can be used in E-Learning for *training* as well as for *assessments*. Self-assessment can be seen as an instrument for training – users get elaborate feedback for their answers and are invited to try again.

In the *training/self-assessment mode* users get feedback after completing the answer or while composing it. The feedback always takes into account all components collected up to

Figure 1: Snapshot of a SET session for answering the SET "What is a parser?"

that point as well as the user's undo or redo actions. The user is allowed to undo a decision as often as he likes. This way finding the right answer is a question of either knowing it or following the hints in the feedback.

In the *assessment mode* the user gets a number of points credited. The points total is compiled the same way the feedback is created. Depending on the answer fragments chosen by the learner, and on their order, the points total will be computed. It is also possible to chain several SETs one after another[5], collect the credits collected in each of them, and present the grand total at the very end.

The user can be allowed to use the undo button in different manners. Three settings are possible:

- The undo button can be used as often as the learner wants but each use is logged in the background.
- Each use of the undo button results in a deduction of a certain number of points, and its use is logged.
- The use of the button is allowed only a preset number of times – if the user tries to undo more often, the button is disabled.

That way tutors can track whether the student arrived at the answer by merely trying out all possible continuations.

4 How to create a SET

What does an author have to consider when creating a SET? First, he has to decide which answer elements the user can choose from at any given step. Second, he must make sure that any of the answer components offered as a choice at a given step will contribute to a well-formed sentence only. Finally, helpful and syntactically well-formed comments have to be defined for any of the possible answers.

What the presentation of a SET ultimately boils down to is running a Finite State Automaton (FSA), with answer components as states

[5]SETs can be linked in linear or network like fashion via HTML links or followups in the comments.

Figure 2: Snapshot of an intermidate result while answering the SET "What is a parser?"

and user choices as input. This is done by a Prolog program as the back-end for a single SET. As input it takes the SET specific Prolog automaton, the path up to now, and the current choice of the user. As output it creates the new current answer, the new list of elements to continue, the preview, comments, paths and points.

The author of a SET has thus to write a (potentially large) FSA. This is a tedious and error-prone task. How can this be done efficiently and reliably?

4.1 The machinery behind a SET

Developing the automaton normally starts with the author writing a number of possible correct and incorrect answers, in a way similar to the development of an ordinary MC. The author then marks where these sentences could be split into fragments. Splitting must allow the combination of various sentence fragments from different sentences in a way that only well-formed passages result. To limit the number of such combinations the author can define constraints that explicitly include or exclude certain combinations.

To increase readability, answer fragments that are of the same syntactic type can be collected in *boxes*. It is, however, advisable to create distinct boxes for correct fragments, wrong fragments, and indifferent fragments of the same syntactic type; this makes the design of complex automata considerably easier. Each box has an ID, in-going and outgoing boxes[6], information concerning specific constraints on allowed combinations, and (positive or negative) credits the user will collect when choosing this element. Boxes are linked by vectored edges to create a number of paths through the answer fragments, each one of which will define a complete and syntactically well-formed sentence.

Splitting answer sentences into fragments that can be combined freely creates, of course, a large number of potential answers (in fact,

[6]Except start boxes – no in-going box – and the boxes at the end of a sentence path – no outgoing box.

Figure 3: Snapshot of the finished SET session for answering the SET "What is a parser?"

a potentially infinite number). It would be clearly impossible to write individual comments for each of these answers. We overcome this obstacle by generating comments, semi-automatically in some cases, and fully automatically in others. The semi-automatical creation relies on the fact that each answer fragment can be rated according to its correctness and relevance for a given question. It is relatively easy to attach, to a limited number of "strategically important" answer fragments, comment fragments specifying in what way they are (in)correct and (ir)relevant. We then have SET collect the comment fragments of all answer fragments chosen by the learner, and combine them into complete and syntactically well-formed comments that refer to the individual parts of an answer and point out superfluous, missing, or wrong bits, in any degree of detail desired by the author. We can even generate comments on the basis of arbitrarily complex logical conditions over answer fragments, thus identifying, among others, contradictions in answers. That way we can generate a potentially infinite number of comments on the basis of relatively few comment fragments. This is the semi-automatic creation of comments, taking into account the local properties of an answer path. We also allow the fully automatic creation of comments that take into consideration the global properties of answer paths. Thus the fact that a learner used the undo button very often in various places, or took a very circuitous way to arrive at his answer, may be detected by measuring global values of the answer path and can then be commented upon automatically[7]. For a detailed documentation see (Brodersen and Lee, 2003).

4.2 Developing a sample SET

Clearly the author of a SET must be supported in the design of even moderately complex FSAs. To this end we developed an authoring tool called *Satzergänzungstest-Ersteller-Interface* (SETEI), a Java application with a GUI. It uses

[7]Resulting in comments like "You used the undo button way too often." or "Correct but your answer could have been much shorter", etc.

a text-based format for saving data and has an export function to create the FSA. Figure 4 shows the final stages in the development of the SET *"Was ist ein Parser?"* ("What is a parser?") used as example in section 3.2.

The box in the left upper corner is the start box, containing the question. Boxes 1, 2, 3, 4, 6, 7, 8, 9, 13 are answer boxes containing answer fragments. Boxes 10, 11, 12, 14 are comment boxes containing comments for complete answers or certain combinations of answer parts (box 14).

One of the boxes, box 14, is selected, and inside this box the text element 72 is selected. As the boxes offer limited space the full text of a selected element is shown at the very bottom of the window. Here we can also see the box number, fragment number, and the credits attached to the selected answer fragment. These credits can be used, in assessment mode, to grade the answer. Creating, filling, and modifying boxes is a matter of a few clicks.

The possible answer paths are represented, obviously, as vectored edges between boxes. Each path must end in a comment box.

- Two paths contain three boxes – 1→8→9 and 1→2→7
- Two paths contain four boxes – 1→2→3→7 and 1→2→6→13
- One path contains five boxes – 1→2→3→4→7

Possible answers in the above example may thus consist of three, four or five parts. Since each answer box contains at least two text elements this automaton defines many more answers than there are paths. On path 1→2, for instance, the user can combine each element in box 1 with each element in box 2. Connections between boxes are created or deleted by simple dragging or clicking operations. Whenever a circular connection is created, even an indirect one, the user is asked whether this is what he really wanted to do.

The top menu in the window contains the various tools for the manipulation of boxes. Thus, to see all text elements in one box plus all the in-going and out-going boxes and the constraints for elements, the author may use the box browser *Ansicht* (view). The browser presents a magnified view on the given box with additional functionalities to edit the box content. The user can also zoom out and see the bare structure of the entire FSA, without box contents, can select sub-parts of the automaton and try them out in isolation, etc.

To allow intermediate feedback, comment boxes may be placed in the middle of the FSA (such as, in this SET, comment box 14). All answer paths end with a comment box to give feedback after creating a complete sentence.

5 Where to use SET

5.1 Where we use SETs

Since winter term 2003/2004 we use SETs at our institute as a training and self-assessment tool in introductory courses on CL. They are often used as final element in learning units intended for self-study by students. These learning units each cover one particular aspect of Computational Linguistics that may be unfamiliar to part of the audience (such as regular expression, tokenising, tagging or parsing). They are organised around Problem-based Interactive Learning Applications.[8] While simple skills can be tested with standard MC methods, for more general and more abstract types of knowledge SETs turned out to be a much better solution. Any type of question that would, ideally, require a free form answer can be turned into a SET. These are definitional questions ("What is a parser?") as well as a questions requiring comparisons between concepts ("How does a parser differ from an acceptor?") and the description of procedures ("What are the processing steps of a transfer Machine Translation system?"). It is important that SETs can determine, and comment upon, non-local properities of answers. Thus a SET can detect contradictions between different parts of an answer, or a wrong sequencing in the description of processing steps (say, putting tokenising after parsing), or repetitions, all of which may occur in parts of an answer that are arbitrarily far removed from each other.

[8] See (Carstensen and Hess, 2003) for more information.

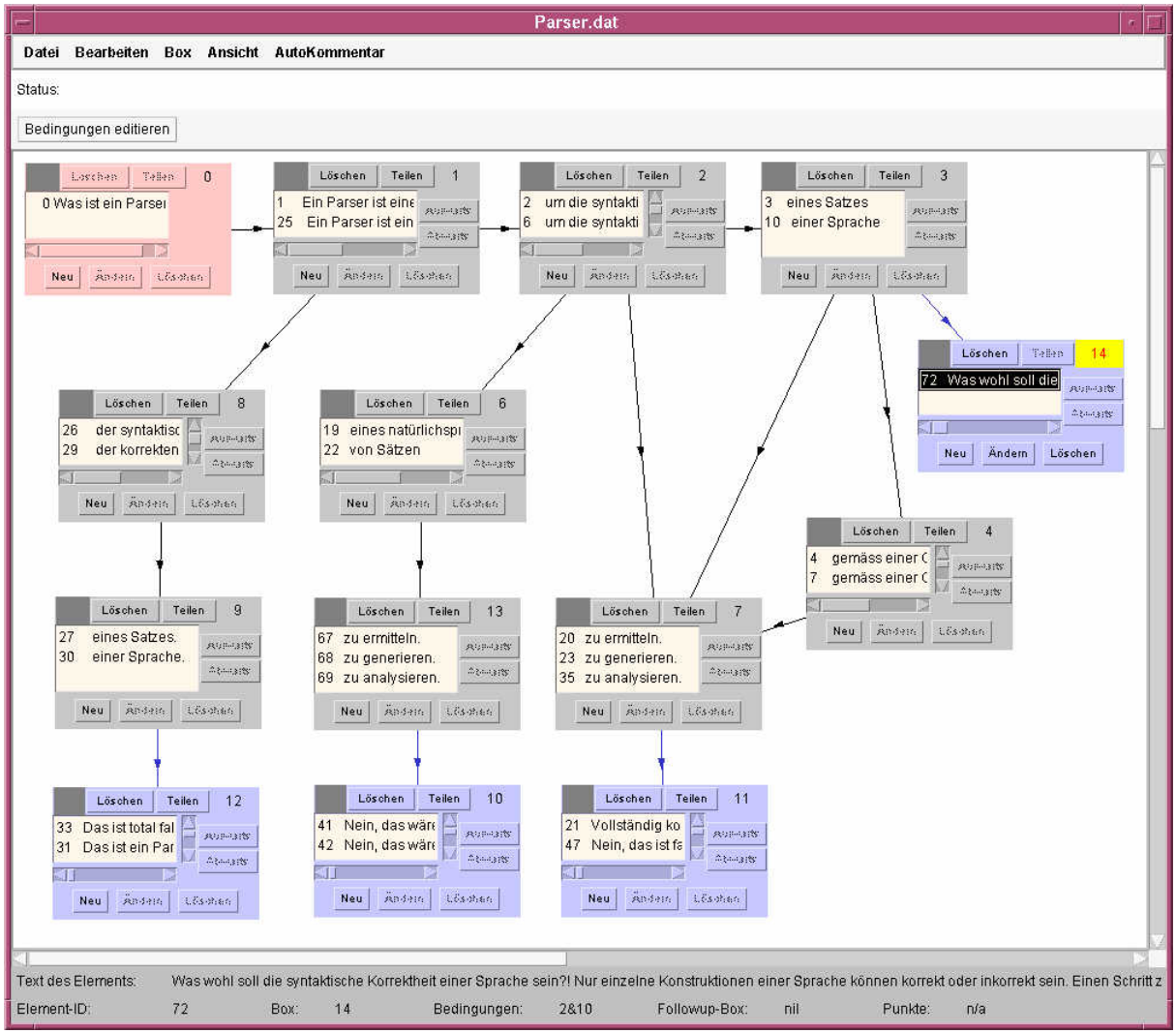

Figure 4: SETEI session for creating the SET "What is a parser?"

5.2 Real Examples of SETs

SETs have been developed mainly for the introductory classes in Computational Linguistics at Zurich but new tests for more advanced courses are under development. Since classes are taught in German, all SETs are in German, too.

Students can access SETs in two ways:

- As most SETs are used in Learning Units students will encounter SETs for the first time when they are working their way through the Learning Units.
- When preparing for exams students want to have random access to SETs. For this reason all SETs ever developed are accessible via one big collection, our *Setcenter*.

The *Setcenter* `www.cl.unizh.ch/ict-open/satztest/setcenter.html` offers a check-box list to create a customised web page containing a short introduction to SETs, help for using them, and a list of links to the chosen SETs. For a first look at SETs the page `www.ifi.unizh.ch/cl/ict-open/satztest/`, with pre-defined examples from outside the field of Computational Linguistics, may also be useful.

Most of the SETs we developed ask questions about the basic concepts and terms of the field. Some examples are listed in table 1.

In some case we also "abuse" SETs to function itself as authoring tool with feedback facilities. In one case students are asked to

Intro to CL 1	Intro to CL 2
CL	Extension / Intension
Linguistics	Propositions
Morphology	Presuppositions
Semantics	Axioms
Parsing	Modus Ponens
FSA	Lambda Abstracting
Generative Power	
Types of Ambiguity	
Indexing	
Information Extraction	
Information Retrieval	
Machine Translation	

Table 1: SETs in the introductory lectures for CL

write specific rules for a chunking grammar. In a SET, they get a set of rule elements to choose from (pre-terminal and terminal categories, parentheses, Kleene star, etc.) and have to combine them, step by step, creating a grammar rule in the process. If their choice of a symbol is completely off track (such as a grammar rule beginning with a closing parenthesis) they are warned right away. Otherwise the structure of the completed rule is commented upon. If the rule is not correct, users are sent back to the beginning. Otherwise they are sent to a subsequent SET, with a more demanding task. That way, by chaining SETs, we teach them to write increasingly complex chunking rules, under close guidance of the system. This turned out to be a very promising use of SETs.

5.3 Use of SET in other topics

The question arises whether it would be possible to use SETs in fields other than CL. In general, in all fields where short textual descriptions are the best way to answer questions, SETs are a good way to automatise training and testing. SETs are of particular interest to the Arts and Humanities, but the Medical Sciences might also be a field that could benefit form SETs (for instance, a picture is presented and the user is asked to describe what seems important or abnormal).

6 Conclusions

In training or assessment situations where correct answers to questions do not consist of one (or a few) isolated items (words, numbers, symbols) but where a complete description in natural language is required, and when human tutors are not available, SET is the right tool to use. It allows to simulate, to some extent, the detailed comments to individual aspects of an answer that make human tutors so valuable.

While SETs are great once they have been written, the process of authoring them is still painful, demanding, error-prone, and thus extremely time-consuming. We will need authoring tools that allow a top-down kind of design for SETs, with stepwise refinement of the code and on-the-fly testing of selected parts of the FSA, instead of the low-level design process used now. It would also be very useful to have programs that work, bottom-up, from possible answers to FSAs, by automatically identifying common phrases in answers and collecting them in boxes. We developed such a system and found it very useful but its grammatical coverage is too small to make it viable in practice. The automatic creation of terminological variations in potential answers, by accessing on-line lexical resources, will be another feature that might make life easier for test developers. We continue work on all of these lines of research.

7 Acknowledgements

Our thanks go to Sonja Brodersen and David Lee, who developed the SET and SETEI environments, to Esther Kaufmann, who created most of the existing SETs, and to Kai-Uwe Carstensen for valuable feedback on the results.

References

Sonja Brodersen and David Lee. 2003. Dynamisches Multiple-Choice mit Satz-Ergänzungstests. Dokumentation zum gesamten Satztestprojekt. unpublished, December 2003.

Jill Burstein. 2003. The e-rater scoring engine: Automated essay scoring with natural language processing. In M. D. Shermis and J. Burstein, editors, *Automated essay scoring: A cross-disciplinary perspective*. Lawrence Erlbaum Associates, Inc., Hillsdale, NJ.

Kai-Uwe Carstensen and Michael Hess. 2003. Problem-based web-based teaching in a computational linguistics curriculum. *www.linguistik-online.de*, 17(5/2003).

European Ministers of Education. 1999. The bologna declaration of 19 june 1999. www.bologna-berlin2003.de/pdf/bologna-declaration.pdf, June 1999.

T. K. Landauer, P. W. Foltz, and D. Laham. 1998. *Introduction to Latent Semantic Analysis. Discourse Processes*.

Donald E. Powers, Jill Burstein, Martin Chodorow, Mary E. Fowles, and Karen Kukich. 2001. Stumping E-Rater: Challenging the Validity of Automated Essay Scoring. *GRE Research, GRE Board Professional Report No. 98-08bP, ETS Research Report 01-03*.

Theodor Rütter. 1973. *Formen der Testaufgabe. Eine Einführung für didaktische Zwecke*. C.H.Beck.