Designing an Introductory Course to Elementary Symbolic Logic within the Blackboard E-learning Environment

Frank Zenker[1], Christian Gottschall[2], Albert Newen[3], Raphael van Riel[3], and Gottfried Vosgerau[4]

[1] Lund University, Sweden
frank.zenker@fil.lu.se
[2] University of Vienna, Austria
[3] Ruhr-University Bochum, Germany
[4] University of Düsseldorf, Germany

Abstract. We report on the design of a blended-learning course in elementary symbolic logic. Challenges and solutions pertaining to the Blackboard e-learning environment (Blackboard Academic Suite Release 8.0) and a customized Gentzen-style proof checker are described. The purpose is to provide orientation for those in the planning stage of similar projects.

Keywords: Blended-learning, Blackboard, proof checker, calculus of natural deduction, online assignment, automatic grading.

1 Introduction

Social, political and economic factors have forced an increase in flexibility and efficiency of higher education. Students regularly pursue part-time employment. Additional constraints arise from early family situations. Others experience limited mobility because of handicaps or long-lasting illness. Under these constraints, a self-paced and internet-based learning model is often perceived as a quasi-natural solution. For an overview of computer-assisted learning solutions, see [6] and, particularly with respect to logic, [4].

When it comes to teaching logic online, best praxis examples are rare. Off-the-shelve solutions are either unavailable or not readily compatible with the e-learning system favored by one's university. To avoid "reinventing the wheel," we briefly report experiences gained when designing an introductory course to elementary symbolic logic at Ruhr-University Bochum, Germany. The course was adapted to the Blackboard e-learning environment as well as a customized natural deduction proof checker. It currently "runs" in the third year, and may be considered stable.

2 Design Issues

2.1 Symbolism

The most important factor in the successful deployment of course-material is the display of logical symbols, especially for online tests. Students will access material

P. Blackburn et al. (Eds.): TICTTL 2011, LNAI 6680, pp. 249–255, 2011.

from various locations and machines. Normally, operating systems and software vary; special characters tend to be variably interpreted and displayed.

To guarantee a correct display without creating image-files, we employed both standard symbols and pursued a "type-writer solution" (below). The script, available in PDF-format, used standard symbols to train students in the most common set of logical symbols. For other formats (HTML for tests or ASCII in chats and email), we used the type-writer solution. Conveniently, it relies on symbols which are *cognitively available*, i.e., printed on the keyboard.

Table 1. Logical symbols

Standard Symbol	Typewriter Symbol	Meaning	Example
$\wedge$	&	Conjunction	*p* & *q*
$\vee$	v	Disjunction	*p* v *q*
$\neg$	~	Negation	~ *q*
$\rightarrow$	->	Conditional	*p* ->*q*
$\leftrightarrow$	<->	Biconditional	*p* <->*q*
$\forall$	A	universal quantification	*Ax F(x)*
$\exists$	E	existential quantification	*Ex F(x)*
$\equiv$	=	Identity	*x* = *y*
$\vdash$	|-	is derivable from	*p* |- *p*
$\models$	|=	entails semantically	*q* |= *q*

Besides the cognitive advantage, this solution saves coding time. Moreover, in large part, the symbolism could also be used to code proof exercises (see Section 2.3).

2.2 Training, Testing and Grading of Online-Assignments

Each week, students took a test within the e-learning environment, consisting of eight to twelve test items. The test had to be submitted within four days after assignment and could be taken only once. The correct solutions, along with the automatically calculated grade, became available to students only after the submission deadline.

For each test, a "pre-test" featuring a roughly equal number of similar items was made available (training runs). This was slightly easier and could be taken *ad lib*. Users received pre-coded feedback upon completion. Such feedback included the correct solution and an explanation why it is correct. In multiple choice tasks, the false choices were also explained.

Design-wise, severely debugging a test before deployment is indispensable, as each faulty test item which goes unnoticed will require additional work to correct grades.[1] As a rule of thumb, the test-designer is least likely to find *every* mistake. In our case, three colleagues provided independent bug reports.

Pre-tests were reportedly useful for appreciating the *kinds* of questions on the test. In fact, we received complaints when tasks differed in kind (which was the exception). Generally, students were unlikely to under-perform on the test when compared to their pre-test score. We see a positive (learning) effect here, insofar as taking the pre-test successfully should reduce anxiety when taking the test.

Test-templates native to Blackboard are of fairly limited use. They seem to be conceived primarily with natural language manipulation in mind, and have to be adapted creatively. For example, the *free text gap* template can be useful as a truth table assignment (place the variable which codes the gap in a table cell), or a semantic proof (with the letters T and F, for 'true' and 'false', to be filled in). Likewise, the *pull-down menu* template allows construction of simple formula derivations, transformations and, generally, any ordering task. The *Yes/No* template proved helpful when testing the mastery of definitions. Expectably, designing *multiple choice*-items is easiest in this environment.

Challenges arose when having formulated the task in an unclear manner, moreover from users' spelling errors and, in complex formulas, from users placing spaces in an unsystematic manner. Notoriously, Blackboard can neither "ignore" spaces nor automatically respect commutativity. For example, in a formula input-task such as 'State the formula described by the following truth table, using only ~ and & next to the propositional variables p and q', the designer will have to code not only '$\sim p \& q$' and '$q \& \sim p$' as correct solutions, but also "blanked variations," such as '$\sim p$ & q' (with spaces between '$\sim p$' and '&', '&' and 'q'), etc.

Blackboard is moreover limited to a maximum of 20 correct solutions per test item. Therefore, we often explicitly demanded *not* to use spaces (making the input less readable) or resorted to coding with pull-down menus. On the (de)merits of Blackboard, see [1].

If a test is well conceived, Blackboard's grading function reliably indicates a student's test performance. When required, the scores automatically calculated by the system can be overridden manually. Generally, grades have to be entered manually for essay questions and natural deduction tasks (which were the minority of the tasks we prepared; see below).

2.3 Automated Check of Gentzen-Style Proofs

Students were required to compose derivations within a Gentzen-style calculus of natural deduction, e.g., 'Derive q from $\sim p$ and p v q' (according to defined introduction

[1] We can report two such occasions at the end of the first third of the course, upon which debugging by a third party was adopted. In the error case, the course designer made the current test unavailable for users that had not taken it already, created and deployed a second version (V2), then corrected the students' grade book entry for the buggy version and retained them in the grade book alongside the revised version. If the "buggy" V1 shall be deleted without having V1 students retake the test, the designer will have to manually enter their V1 grades into the V2 test results – time-wise, another black hole.

and elimination-rules). We introduced an adapted version of the calculus described in [2]. Automatizing this task proved to be the project's foremost technical challenge. It could only be met incompletely after custom-fitting an interface between the proof checker application and Blackboard had to be excluded for budget reasons. The proprietary Blackboard code was the most important factor hindering a full integration, because the prospect of having to backward-engineer it made the extent of the project unforeseeable.

The solution we developed mimics integration. It uses *building blocks*, a Blackboard software-enhancement. This allows sending a time and a user ID along with the data that users enter into a form on the input page of (what we refer to as) the "proof checker." This page was requested from the server by means of a Blackboard internal link; the derivation task is coded as part of this link.

For example, the snipped '…task=~p+%26+~q,~%28p+v+q%29…' generates the test-item: 'Prove that *~p & ~q* is derivable from *~(p* v *q)*'. The conclusion is written first. What follows after a comma is a premise. '+' codes a space, '%26' codes '&' while '%28' and '%29' signify a left and a right bracket. Conveniently, a subset of the typewriter symbols (see table 1, above) can be reused both for coding the derivation task and for completing it.

The proof checker's results were not available to students. They could only be accessed by their tutors through a password protected area. Based on, but not determined by the program's responses (e.g., "Not derived in a rule compliant manner, error in line X"), grades were assigned and manually entered into the Blackboard grade-book. A natural future development is to fully integrate the proof checker, so results are automatically transferred to the grade book.

A noteworthy feature is the *syntax validation function*. It is implemented as a button reading 'check formula' below the proof-checker's input form. The form consists of ten rows of four free-text fields (dependent premises, formula, derivation rule applied, lines used); clicking a second button generates additional rows, line numbering is automatic. Before submitting a proof, students can have their input checked to "weed out" mistakes.

For example, in response to a syntactically faulty line such as '*p ~* v *q*', the proof checker reports a general error, e.g., 'Syntax error in line 3'. This extends to less obvious mistakes, e.g., 'The Rule for OR introduction cannot be applied in this way. Note that the order of lines to which the rule is applied is relevant'. Through this feature, we could keep submissions from containing mistakes, while allowing errors.

The proof checker is a custom-adaptation of a program running on Christian Gottschall's website "Gateway to logic" [3]. It performed extremely well. Unlike the blackboard native input fields (see above), the proof checker does ignore spaces.

2.4 Programing Considerations

The proof-checker program has been around since 1992. Based on the calculus of Lemmon [5], it reads a text file containing a derivation, then reports whatever it finds wrong within the respective derivation. Wrapped in a CGI layer, the program later became part of what is now the "Gateway to Logic".

Adapting the program to the course's calculus was a fairly simple affair. However, in the future, we would prefer not to code the parser manually, but to use a "compiler

compiler" (e.g., yacc or javacc). This would have significantly accelerated changes in syntax. Further, using a higher-level programming language (i.e., higher than C) – which, for performance reasons, was not an option at that time – will considerably reduce the time needed for adapting and testing.

On the programming side, the main challenge was the seamless integration of the proof checker into Blackboard's user-flow and user-experience, while minimizing the possibility of manipulation. After all, the proof checker is used for submitting homework, and can be used for exams. We could re-use the existing CGI wrapper from the "Gateway to Logic," although the level of required adaptation was higher than with the proof checker.

In contrast to the "Gateway" proof-checker, the Blackboard version required the implementation of statefulness. States were: (1) the input form with the default number of blank proof lines; (2) the filled-in input form with an additional proof line (being requested by the user); (3) the form after a quick initial input check; (4) the checked, and commented, proof (for practice runs); and (5) a confirmation of the successful submission of a derivation. Furthermore, at each stage transformation, user credentials, and lifetime information for the respective request had to be generated, signed, and properly passed over.

For projects without legacy software, we recommend using as high-level a programming language as possible. Especially explicit list and set processing capabilities come as a plus, and do not necessarily require (things as fancy as) Prolog or LISP. For a number of younger projects, we can report excellent results obtained with Java.

Further, it is recommendable to delegate as much routine work as possible (here: mainly CGI issues) to some open middleware (e.g. class libraries), rather than addressing these issues "by hand". This holds especially for language processing: Do define the syntax of the logical language, and the calculus as a whole, in some established meta-language (BNF, EBNF), then have it parsed automatically. With recent Java-based projects, JavaCC was our tool of choice.

2.5 General Considerations

When designing the course material, it is good to keep in mind that at least one entire session will be "lost" explaining to students the details of using the e-learning tools. Especially for a pure online version of the course, it is advisable to meet/contact students in advance. Currently, at introductory level, the majority of students are unacquainted with e-learning environments.

During online-based tutorials, questions related to these tools arose frequently, thus reducing the time allocated to discuss the session's content. Fortunately, this did not result in grave problems. The loss of time, it seems, was counterbalanced by the fact that students were enabled to find answers themselves, using audio/video-recordings of lectures, as well as pre-tests, an online glossary, and the course script.

2.6 Final Exam

The final exam was a classic pen and paper test. However, most of the exam-questions were in the style of the online tests. Conducting a final exam online did not appear viable for various reasons. Generally, tests should not be taken at home, as this

not only encourages cheating, but also increases the risk of computer system or internet connection failures. Thus, a room with a sufficient number of workstations connected to a server strong enough to handle a high number of quasi-synchronous users is required. These constraints, however, could not be met.

In the future, we plan a mixed procedure by using EvaExam.[2] The program allows generating a paper exam featuring barcodes and a defined layout. The completed exams can be scanned and automatically evaluated. This, of course, only works for multiple-choice tasks. Other types of tasks will have to be evaluated manually. The grade is noted on the paper and then read out by the program. Thus, the tasks from the online tests can be largely (re)used for a pen and paper exam and be evaluated automatically.

3 Lessons

Although the majority of students reported no problems with handling Blackboard, we had to learn that our enthusiasm about the e-learning project was not generally shared. In fact, some students were downright adverse to new media, and dropped the course as soon as they noticed it employed e-learning techniques. Principal criticism also arose, because some argued that e-learning offered an excuse or pretext for not hiring additionally staff, thus reflecting a false development of the university system.

During the first two weeks, we catered for students' differential affinity to new media by providing ample opportunity for individual help with anything from obtaining a user name to arranging a wireless network connection on campus. Fortunately, few needed it, but we think that some might have performed worse without it. Therefore, in the first few weeks, we offered one special online tutorial dedicated to technical questions only.

As stated, course material was made available as a script, and supported by a video recording of the weekly plenary meeting. Nevertheless, students benefited greatly from the weekly online tutorials (one tutor for 15 students). With this important instrument the online course ran rather smoothly. The only part of the course that students truly needed support with was the calculus of natural deduction. So, we resorted to introducing the convention that an online-course (which normally knows no real "face-time") features three meetings. The first is used to introduce the technical background. After two thirds of the semester, one meeting is spent on introducing the rationale of the calculus of natural deduction, before presenting its details online. Finally, the last meeting is used for the exam. With this strategy the course has proven to be extremely useful.

We recommend collecting material from previous courses well ahead of time. Much of it will prove useless, because it cannot be transferred online without major change. Therefore, seek lots more than you think you will need. Having a script ready is a plus. However, it will likely have to be revised to accommodate the online exercises. Generally, most assignments which were formerly graded by humans need substantial revision. Most will have to be built from scratch.

Expect at least one full hour for building one test-item, no matter how simple it looks. Allow two hours for the first twenty items. On those, half your time will be

[2] EvaExam by Electric Paper, http://www.electricpaper.biz/

spent learning what you cannot do. If one subtracts a final exam and perhaps the first session, then, for a 14 week course featuring one weekly pre-test and one test with, on average, 10 items, you face 140 hours of build-time – merely for implementing the tests, not for finding your material. As stated above, building is one part, severely testing the other. One hour of debugging is reasonable per ten test items and reviewer, on pains of spending thrice the amount on corrections.

If reviewing what goes online may count as essential, performing back-ups should count doubly so. Blackboard comes equipped with an easy to use back-up tool. This generates a compressed file of the entire course or parts (e.g. tests) for local storage. In the worst case, a course can be completely restored from this file. A weekly backup should be the minimum standard. As it takes five minutes but might save your neck, a daily backup can be recommended.

Readers interested in details are encouraged to contact the corresponding author.

Acknowledgements. We thank our tutors, Natalia Eberle, Uwe Hunz, Hans-Joachim Höh, David Neugebauer, and Kathrin Braungardt and Jens Riedel for tech-support.

References

1. Coopman, S.J.: A critical examination of Blackboard's e-learning environment. First Monday 14(6) (2009), http://firstmonday.org/htbin/cgiwrap/bin/ojs/index.php/fm/issue/view/291
2. Forbes, G.: Modern Logic. In: A Text in Elementary Symbolic Logic. Oxford University Press, Oxford (1994)
3. Gottschall, C.: Gateway to logic. A collection of online logic tools (2011), http://logik.phl.univie.ac.at/~chris/gateway/formular-uk.html
4. Huertas, A.: Teaching and Learning Logic in a Virtual Learning Environment. Logic Journal of the IGP 15(4), 321–331 (2007)
5. Lemmon, E.J.: Beginning Logic. Chapman and Hall, London (1987)
6. Mayadas, A.F., Bourne, J., Bacsich, P.: Online education today. Science 323, 85–89 (2009)