HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Electronics, Communications and Automation

Matti Harjula

Mathematics exercise system with automatic assessment

Master's Thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Helsinki, March 28, 2008

Supervisor: Professor Heikki Koivo

Instructor: Antti Rasila, Ph.D.

HELSINKI UNIVERSITY OF TECHNOLOGY

ABSTRACT OF THE MASTER'S THESIS

Author:	Matti Harjula	
Name of the Thesis:	Mathematics exercise system with automatic assessment	
Date:	March 28, 2008	**Number of pages:** 75 + 19
Degree Program:	Degree Program of Automation and Systems Technology	
Professorship:	AS-74 Control Engineering	
Supervisor:	Prof. Heikki Koivo	
Instructor:	Antti Rasila, Ph.D.	

Automating mathematical exercises may provide both quality and efficiency benefits. This thesis examines the use of a system meant for this purpose and then considers the features such a system should have and how some of them can be implemented. Features such as integration to Single-Sing-On (SSO) systems and rendering of mathematical content in varying environments are discussed and then implemented for the chosen system.

Statistics from a test course using that system is analysed both from the general level and from the point of view of the answer input process and its problems. Solutions for problems observed with inputting the answers will be suggested and some corrections are implemented.

Some of the possible exercise types are examined, and observations on how one should construct them are given. Best practises on how generic exercises should be implemented with a system of this type are discussed.

Keywords: thesis, CAA, mathematical exercises, automatic assessment, grading, STACK, Maxima, SSO

TEKNILLINEN KORKEAKOULU DIPLOMITYÖN TIIVISTELMÄ

Tekijä:	Matti Harjula	
Työn nimi:	Mathematics exercise system with automatic assessment	
Päivämäärä:	28.3.2008	**Sivuja:** 75 + 19
Tutkinto-ohjelma:	Automaatio- ja systeemitekniikan tutkinto-ohjelma	
Professuuri:	AS-74 Systeemitekniikka	
Työn valvoja:	Prof. Heikki Koivo	
Työn ohjaaja:	FT Antti Rasila	

Matemaattisten harjoitustehtävien automatisointi voi johtaa tasaisempaan laatuun ja kustannustehokkuuteen. Tässä diplomityössä tutkitaan yhden tähän tarkoitukseen suunnitellun järjestelmän käyttöä ja pohditaan yleisesti sitä, minkälaisia ominaisuuksia tällaisella järjestelmällä tulisi olla ja kuinka ne voitaisiin toteuttaa. Joitakin ominaisuuksia kuten kertakirjautumisjärjestelmiin integrointi ja matemaattisen sisällön esittäminen vaitelevissa käyttöympäristöissä esitellään tarkemmin, ja toteutetaan valittuun järjestelmään.

Kyseisen järjestelmän koekäytöstä kerättyä dataa analysoidaan sekä yleisellä tasolta että erityisesti vastausten syöttöön liittyvien ongelmien kannalta. Havaittujen syöttöongelmien pohjalta esitetään parannusehdotuksia ja korjauksia.

Joitakin mahdollisia harjoitustehtävätyyppejä tutkitaan toteuttamiskeinojen ja niihin liittyvien seikkojen kannalta. Yleisimpien harjoitustehtävien toteuttamisen kannalta hyödylliseksi havaivaittuja käytäntöjä esitellään tarkemmin.

Avainsanat: diplomityö, CAA, matemaattiset harjoitustehtävät, automaattinen arviointi, arvostelu, STACK, Maxima, SSO

Preface

The process of writing this Master's thesis has taught me many lessons that I did not expect when I began the process. Those lessons will no doubt be of use in the future and I warmly thank my instructor Antti Rasila for his guidance through them. Professor Heikki Koivo should also be thanked for delegating the supervision of the thesis to Kai Zenger, who provided the perfect counterbalance for my technical bias and thus kept the thesis in better balance.

I wish to thank Simo Kivelä and Jarmo Malinen for hiring me to work in this project back in 2004. Working at the MatTaFi project has been an experience that I will not soon forget. The fact that the project provided a way for me to hone my technical skills while studying more theoretical ones was very important for keeping me motivated to keep studying.

I am also grateful to all the people who took part in the testing of the system at the TKK Institute of Mathematics. My special thanks go to Juhana Yrjölä, for filtering the raw feedback from Malinen.

Finally, I would like to thank the people sharing my workroom, Mikko, Toni, and Igor for tolerating my existential angst while writing this thesis, and yes, Igor, I will shut up now...

Otaniemi, March 28, 2008

Matti Harjula

Contents

Abbreviations

AIM	ALICE Interactive Mathematics, AIM software was renamed in version 3.0 to AiM that stands for Assessment in Mathematics.
ALICE	Active Learning In a Computer Environment, a system developed by Norbert Van den Bergh and Theodore Kolokolnikov at the University of Gent in Belgium.
CAA	Computer-Assisted Assessment or Computer Aided Assesment.
CAS	Computer Algebra System.
CAT	Computer Adaptive Testing.
CSS	Cascading Style Sheets, a language used to describe the presentation of a document writen in a markup language. Typically used with HTML.
GNU	The free software project initiated by Richard Stallman. "GNU's Not Unix".
GPL	(GNU) General Public License, one of the standard open source licenses.
GUI	Graphical User Interface.
HTTP	Hypertext Transfer Protocol, basic transfer protocol used for transfering data in the Internet.
LAMP	Linux, Apache, MySQL, and PHP. One of the most common web platforms to build open source web applications.

LDAP Lightweight Directory Access Protocol is a common method for accessing directory services like databases over a network. It is especially common when interfacing varying user account management systems with other systems.

MAMP Mac OS X, Apache, MySQL, and PHP.

MVC Model-View-Controller, basic design pattern used in software engineering. It divides an application into three separate pieces.

PDF Portable Document Format.

PG Problem Generating language, used for defining problems in WeBWorK. Closely related to Perl.

PHP PHP: Hypertext Preprocessor, is an scripting language commonly used for web applicatios.

POSIX Portable Operating System Interface for UniX.

RMS Root Mean Square is a measure generally used to measure the quality of a models fit to a data set. Typically it is defined as $e_{rms} = \sqrt{\frac{1}{n}\sum_{i=1}^{n} e_i^2}$, it can also be used to reference to a free soft-fare activist Richard M. Stallman.

SAML Security Assertion Markup Language is an XML standard for exchanging authentication and authorization data.

SOAP Simple Object Access Protocol is a protocol for exchanging XML-based messages.

SP Service Provider.

SSL Secure Sockets Layer is a cryptographic protocol that provides secure communications on the Internet.

SSO Single Sign-On.

STACK System for Teaching and Assessment using a Computer algebra Kernel.

SVG Scalable Vector Graphics is an XML markup language for describing two-dimensional vector graphics.

TtH	An application converting TEX-to-HTML.
UA	User Agent, e.g a web browser.
UI	User Interface.
URI	Uniform Resource Identifier.
URL	Uniform Resource Locator.
VLE	Virtual Learning Environment.
VRML	Virtual Reality Modeling Language
XAMP	General term combining LAMP, MAMP and WAMP.
XML	The Extensible Markup Language is a general-purpose markup language.
XSLT	Extensible Stylesheet Language Transformations is an XML-based language used for the transformation of XML documents. Typically used to convert data in XML-documents in more humane form.
WAMP	Windows, Apache, MySQL, and PHP.

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

During the last decade the quality of teaching has become a main topic of interest in the Finnish higher education system and in the European Union in general [1]. At the same time financial constraints have become tighter. Therefore we stand at a situation that requires more quality and more efficiency in teaching. While one cannot do much to improve the efficiency of the classroom education with new technology, it is possible to invest resources in the use of new technology in teaching materials used both inside the classroom and outside it. Improving the quality of the materials has some constraints, as the traditional static exercise and teaching materials tend to have issues with reuse and with the tacit knowledge associated to them.

With new electronic document formats we may embed more information to the teaching materials and even make them interactive. With the Internet as a distribution channel it is easy to share and deliver the material in its purest form. While these newer technologies suite well for basically every type of teaching material from presentation to course books and demonstration tools, the most interesting type of teaching material are the exercises [2] as they can benefit the most from interactive features, addition of meta-data and the use of the Internet to access them. From here onwards we focus on interactive exercises that are shared and used via the Internet.

With classroom teaching materials like lecture notes and presentations we have no problem with reuse. With sufficient information a different lecturer may almost immediately make use of preexisting material and thus save the time needed for compiling such materials. The situation has remained the same for long time as

most basic courses can be lectured directly from ready-made material i.e. the course book.

With exercises the situation is different. There are limitations with reuse because we cannot use exactly the same exercises too often. Students might get tempted to seek out possible answers from various sources. At the same time we need to have exercises that work well together. But the main problem with exercises is that they must be checked. The checking process is expensive and does not scale well with the number of students. Once exercises have been checked it would be nice from the point of view of quality education to give feedback to the students. For reuse it is important that the person choosing the exercises would not need to disassemble them before deciding whether they are suitable. Hence plenty of meta-data about the exercises is needed, at-least a detailed solution. Naturally a reusable exercise of good quality should also contain sufficient documentation for its use. Materials like guided solutions and checklists for grading are slow to reproduce and should therefore be available for use if possible. While good and experienced teachers can immediately see if an exercise is suitable for use in some specific context the tacit knowledge needed for identifying those features in an exercise takes time to build up for new teachers, to avoid situations where an unsuitable exercise is picked blindly there should be enough information about the use of the exercise available.

Parametrised exercises are exercises that have parameters that can be varied [3] to generate new instances of the exercises when needed as well as the solutions and other documentation. Parametrisation increases the reusability significantly, but the problem is that writing and validating these kinds of exercises is extremely time consuming with traditional methods. Naturally, traditional exercises have always been modified by changing some small detail here and there and then solving the problem again. These methods have basically required work to be done every time an new instance is generated and for each time there is some possibility for making errors. For exercises that can be assessed with a computer, such as basic mathematics exercises, it is possible to generate parametric exercises easily. At the same time it is possible to do fast checks for as many of the possible instances as one wishes.

Various eLearning [4] technologies have become mature enough in the recent years to help us provide some of the just mentioned features for our exercises. For some subjects we can even use automatic assessment and feedback methods to avoid or at-least minimise the effort associated with checking and feedback. These technologies also bring in new possibilities that could be of use. But the main point of interest is the fact that these technologies tend to scale well for ever larger groups of students.

Mathematics can benefit greatly from eLearning, since basic mathematics exercises can usually be formulated in such a way that they work well with automatic assessment. Presently, assessment systems for mathematics are still somewhat incomplete and specifying something that is not so basic can require a considerable amount of work. In this thesis the possibility of building more complex exercises in a mathematics assessment system, deploying the system for use at a basic level mathematics courses, and formulating some more complex exercises with it, is studied.

Teaching with these new technologies has been studied from many points of view and in varying scales. Those interested in eLearning as a larger subject may be interested with *Theory and Practice of Online Learning* from Athabasca University [4], while those more interested in teaching mathematical subjects may find papers [2, 5, 6, 7, 8] written by Chris Sangwin, a developer of one mathematical exercise system, more relevant. Data from this thesis has also been included in a research paper [9]. Much information is also available on the Internet, for example, *JEM – Joining Educational Mathematics*[1] network has collected an impressive collection of papers on this subject.

1.1.1 Computer-assisted assessment system terminology

The term Computer-Assisted Assessment (CAA) system will repeatedly appear in this thesis from this point onward. The term CAA system is defined here as a system that:

- Is running on a computer, typically on a web-server serving web-pages to users over some open or closed network.
- Displays exercises to users and receives answers that it will automatically check by using some sort of an grading algorithm. For mathematics exercises the algorithm will most likely be evaluated by a Computer Algebra System (CAS).
- Stores the results of those checks for grading and/or assessment.
- Allows the formulation of new exercises using tools provided by the system i.e. the exercises are not hard-coded. One should also be able to formulate parametric exercises.

[1] http://www.jem-thematic.net/

1.1.2 MatTa and MatTaFi

MatTa Project has been running at the Institute of Mathematics of the Helsinki University of Technology since 1993. In the project various technologies available to teachers for creating electronic study materials have been developed and tested. At the same time, the pedagogical effects of the materials and use and usability of both materials and tools related to them have been studied. Under the leadership of Simo Kivelä the following areas were studied:

- Presentation of mathematical material (for example LaTeX, PDF, MathML...)
- Mathematical animation (VRML, Cabri, LiveGraphics3D...)
- Use of computer algebra systems (Mathematica, Maple, MATLAB...)
- Web based mathematical CAA systems (webMathematica, MapleTA, STACK...)
- Material reuse and storage with material-banks and search functionality (Mahka, Euler 1&2...)

MatTaFi Project is aimed for spreading the knowledge gained from MatTa Project to universities and polytechnics across Finland. It is concerned with similar research as MatTa Project but concentrates more on network based applications that could be shared between the members of the project. Currently there are 5 universities and 5 polytechnics participating in the project. The current project leader Antti Rasila is also the instructor of this thesis and the thesis is done within the project.

1.1.3 Goblin and Trakla

Goblin [10] and Trakla [11] are some of the best known web based CAA systems created at Helsinki University of Technology. They are primarily intended for teaching programming but could be customised for other tasks as well.

Goblin is the system currently used in programming courses arranged by the Department of Automation and Systems Technology. It was initially created for the needs of a C/C++ course but has spread to other programming languages as well. Goblin has been developed in the research group of Information and Computer Systems in Automation.

Trakla is a system used to teach algorithms and data structures. It visualises the way data structures and algorithms work and makes students execute different operations by dragging objects and setting bits in Java applets. Trakla Project has produced several research papers about the pedagogical aspects of computer aided

assessment. Trakla Project runs at the Laboratory of Software Technology. Trakla's usage seems to be limited to a data structures and algorithms course.

Goblin contains its own user management and authentication systems and requires students to remember one more password. But as the use of single sign-on technology becomes more and more commonly used at Helsinki University of Technology it will surely start using it. Trakla has already started to use single sign-on technology as an alternative login method.

1.2 Contents

This thesis describes the structure of a mathematical exercise system, the basic features one should expect from such a system, and how one could build exercises for such a system. The thesis begins with an introduction to the motivation behind these kinds of systems in chapter 2.

Chapters 3–4 describe the basic structure of CAA systems in general and then continues to the specifics of one such system, STACK that was chosen for use in test courses. The main aim of chapter 3 is to provide the reader an overall picture on the structure of a CAA system, understanding which components are needed for such a system is required to evaluate new systems and may also be used as an basis for designing new ones.

Chapter 5 contains data collected from a test course using a CAA system. That data has also been used in a research paper [9] analysing the data from slightly different point of view.

Chapters 6–7 describe features added to the chosen system after initial evaluation. Chapter 6 concentrates on features required for using the system alongside other systems as well as features required for more efficient administration of the system. Chapter 7 focuses on the user-interface changes done to the system, and the reasons behind them.

Chapter 8 gives examples of some of the question types that are possible with the chosen system as well as some guidelines for constructing exercises with the system.

Chapter 2

Motivation

2.1 Time and money

Motivation for investing time and resources to study CAA systems comes from two different factors, the financial and pedagogical. Naturally there are also some other sources of motivation, the most important of them being convenience.

The fundamental question is naturally: Why would one use time and effort to construct an automatic computer-based assessment system instead of using traditional methods?

2.1.1 Scalability & reuse

Scalability is the main benefit of automatic assessment from the financial point of view. The cost of traditional exercises consists mainly of the salary of the people grading them. The number of people depends on the time limits and on the number of answers to be graded. For CAA systems the number of people involved is only about two as we only need someone to create exercises and someone to keep the system running. The number will not rise even when the number of answers to be graded rises, nor do normal time limits matter when the CAA-system can normally grade an answer in a fraction of a second.

Naturally CAA systems will not help in the guided exercises in a class room but they will make it possible to have more exercises with feedback outside of the class room. So CAA systems will never solve the problem of allocating space for exercises but it is possible that CAA systems will lower the time and space needed for exercise sessions as some exercises can be arranged outside the class room.

After a CAA system has been set up it can be used continuously with little or no additional cost, whereas in the traditional exercises one may need to train new

people to grade the same exercises on the next year's course. With parametrised exercises one may even use the same exercises without need to create new ones or to seek suitable replacements.

Long term issues with reuse of exercises

Because CAA software is a complicated system, and it also depends on the client system, one will eventually end in a situation where the CAA system no longer works well enough and must be upgraded. Upgrades may be somewhat expensive. Depending on the backwards compatibility of the CAA system one may need to rewrite some or all exercises to work with the new system. Fortunately, usually the system is usable for quite long time. In particular, systems based on open standards, and especially those based on open source technologies, should remain usable for nearly a decade (in some cases as long as the hardware runs and there are web browsers supporting HTML; that could mean decades). When the system eventually has to be upgraded, the technology has probably advanced so much that the new system offers significantly better functionality.

As an example of a system that has been in use for a long time is the entry level assessment system in Tampere University of Technology [12]. That system has been running for a whole decade and seems to work just well even now. The problem with updating that system is not finding a new system to use but the fact that changing to any other system will change the way things have looked like for so long. Any change will make comparison between results from different years somewhat difficult even if the exercises were identical in the new system.

2.1.2 Pedagogical advantages

From the pedagogical point of view CAA provides the possibility to increase the number of simple exercises and at the same time to provide instant feedback to the student. CAA systems can easily be used to give more exercises and to lure students to do them by giving extra credits. The exercises can be made simpler and the credits scaled accordingly. But from the students point of view a simple exercise might be worth the effort while bigger exercises might intimidate the student away.

Naturally, the increased number of exercises is good for teaching but the biggest benefits come from the data about errors. With a CAA system students do not necessarily check their results too carefully before returning them. One should therefore get a better picture about the student's idea behind the answer. For this reason one should encourage students to submit even wrong answers rather than

not submitting anything. If the system gives a chance to correct a wrong answer, these correction attempts might even give more information.

The most important thing is that the students do the exercises, and when they learn to do those simple ones one can always mix in some harder ones. One can either use simple and fast to solve exercises or exercises that give plenty of credits to lure students to start doing the exercises. Although one should remember that luring students to do exercises with credits in a system that gives instant feedback might drive some people to fall into a "one more credit"-mode which may not always be the best-practise for learning, there must always be a maximum number of credits after which one may not gain any more, otherwise the situation may deteriorate to a competition where students may start to use questionable methods to gain more points.

While a CAA system gives plenty of data about students errors it may not be the optimal way to assess the skills of the students. To assess the skills of a student before choosing what the teacher should emphasise one should use a computer-aided testing (CAT) system instead. CAT systems will give a better picture of the skills but may require much more work to set up. CAT systems will choose the exercises presented to a student based on that students errors on previous exercises thus probing for areas needing improvement.

2.1.3 Convenience

CAA gives the possibility to set the deadlines for exercises to any time regardless of opening hours of school buildings. For example, answers returned on paper could have a deadline at a Friday afternoon because the teacher would like to have a glance at them before Monday mornings lecture. Now the deadline could as well be at Sunday evening and the teacher may check the full results any time he wishes to via the CAA system's web-site. From the student's point of view, the extra time gives a possibility to arrange tight timetables and maybe even a chance to use more time for the exercises. But from both the student's and the teacher's point of view the most important thing may just be that there is no need to go to the mailbox and return/get some papers.

CAA system might also give the student the chance to do the exercises whenever and wherever he/she wants to. The exercises are always available to the student and without a need of writing pretty answer papers the student may as well do the exercises with his/hers mobile phone in a bus.

From a more ecological point of view the elimination of answer papers should also

be a significant benefit as reasonably sized courses can easily generate thousands of answer papers. Of course an important advantage is the removal of those extra papers from ones desk.

2.2 Quality

As pointed out by Finnish National Board of Education [1], the quality of education has been given more and more attention in recently. Next we will study if CAA systems can bring new quality to teaching [13]. If so, these systems could become a way to increase the quality of exercises and therefore of education. Of course, the problem of the definition of quality is still generally lacking, but at least few intuitive definitions apply here.

2.2.1 Quality as repeatability and uniformity

The quality of manufactured items can be defined in many ways, but the most obvious ones require that any two items are equal or identical and that there is no variance nor do they act differently [13]. For a CAA exercise assignment this always holds as every time a student gives the same answer to the same question the system reacts similarly and everyone doing the same mistake will get the same feedback. This kind of a behaviour cannot always be expected from human checking, since as the checking progresses there is no way to make sure that the last paper graded gets the exactly same amount of attention as the first one.

2.2.2 Quality as excellence

One could define excellent exercises in many ways but for a CAA system the exercise itself might not be the main thing. Instead, the checking algorithm would be the part that brings out the excellence. The difference with traditional exercises and CAA exercises is the checking algorithm's capability to store feedback for a countless number of specific errors and to give that feedback equally to all students. What is really important is that as the exercise evolves during the years the algorithm may be expanded to include more and more special cases, and at some point the current students may actually benefit from the information gained from some very eccentric errors made by some other students on a different course several years ago.

Although the feedback could always be generated again when the exercises are manually checked, it is unlikely that one would try to explain what went wrong to all of them. But when using CAA one can simply check the system for errors with

no specific feedback to find out what kind of new errors the students have made this time. If there are a significant number of similar errors one can add a check for them in the assessment algorithm.

Chapter 3

Basic structure of a CAA system

3.1 The parts of a system

From the point of view of the model-view-controller (MVC) architectural pattern [14], one can find some specific parts that are useful in comparing different CAA systems.

The MVC pattern is a common architectural pattern, particularly common with web applications. However, it can be used for other purposes as well. One should remember that MVC is an architectural pattern, not just a design pattern [15]; it is meant for application level design and not for small scale design.

According to the MVC pattern an application consists of three different parts:

M The model part defines the data used in the application, for a CAA system the model would consist of exercises, users and results.

V The view part displays the data and the controls used to modify it, for a CAA system this would include the exercise forms, the user management views and the authoring tools for creating new exercises.

C The controller modifies the model according to the input received from the view and other sources. For a CAA system the controller would take the student's answer and evaluate the grading algorithms defined in the model to calculate the new score to the student. This score would then be updated to the model.

In this thesis we concentrate on web-based systems with mathematical exercises.

Therefore the definitions of these parts are written from that point of view.

3.1.1 The model

The model part is generally the most important part of the system as it defines what kind of data the system can represent. For a CAA system there are many different areas where models do matter.

First comes the user model and the concepts of groups and rights. Does the system have distinct users for students and can one group these users is a question of great importance, as if there are no distinct user accounts for students, there is no way of connecting the scores to a specific student. If there are no groups then the handling of multiple courses on a same system becomes complicated. For running multiple courses on the same system there must also exist some kind of a user rights model so that only some users can access a certain content and others can not. The bare minimum for a CAA system used for giving grades is that there are distinct user accounts for students. Other user, group and rights related features of the data model tend to be useful only when the same installation of the system is used on multiple courses simultaneously.

The second part is the storage of results and answers. Does the system store the answers given by the students, or does it only store the scores? Does it keep all answers or just the last one, and are answers connected to users or are they disconnected to provide anonymity. This is an important question, especially regarding the question on keeping track of common errors and on focusing teaching effort to them. Although the main aim of a CAA system is normally to get the grades for the students one should not ignore the information that can be obtained from the errors made by the students.

Third and the most important part are exercises. Can one group exercises as a series or must one handle them one by one? Can one use the same exercise directly in multiple places or must one copy it? Can one define when an exercise becomes visible and when it closes? The most interesting part is of course the definition of the exercises and their grading algorithms. For example, does one define just one exercise or a template for multiple exercises that depend on some (random) parameters? How does one define what kind of an input the system should expect from the student?

The model for the grading algorithms should be particularly well designed, and it should allow easy formulation of the algorithm. Can one define the algorithm directly as some-kind of an code or must one use some predefined algorithms or test

functions and can one apply multiple tests to the answer? How about tests based on the results of other tests? How large can the algorithm be, does the model limit it?

3.1.2 The view

The standard user interface (UI) needs to display the exercises for the students to answer, the menus for choosing the exercises, and the management tools for teachers to create the exercises and to handle the answers. Extended UIs need methods to display exercises for multiple different courses and methods to handle and analyse answers in more complex ways. For an old CAA system the UI could well have been text based but nowadays the UIs are graphical. For a web based system the UI is naturally rendered in HTML and displayed via the student's standard web browser. However, even a web based system could have its own UI program that displays the UI.

For web based systems the use of the web browser is of course the simplest way, as it means that almost any computer can be used as a client and no extra programs are needed. Naturally, rendering the UI in HTML so that all the different browsers display it in the same way is not so easy. But providing working HTML for the three most important web browser types can be done somewhat painlessly. Internet Explorer, Gecko-based and WebKit-based browsers should not prove impossible in this respect.

The biggest problem for UIs are mathematical exercises and their symbols and equations. For a text based UI it is pretty much impossible as all output would be LaTeX-code, pretty-printed equations or some kind of CAS code and for a GUI things get even more complicated as the presentation must be done with bitmap images or with vector based representations. Bitmap based image representation tends to be slow and bandwidth intensive as it requires images to be rendered and transferred to the student. On the other hand, an image based system will work with any browser as long as the images can be rendered and the resolution is high enough to see all possible details in the formulae, e.g. indices. The biggest weakness in image based representations is that it cannot be scaled with other contents of the GUI and that there is no simple way to modify the image representation after rendering. Vector based representation on the other hand tends to look better and scales with the GUI, but it requires more effort from the GUI to be displayed.

Vector based representations can be divided in to two groups: pregenerated and client side generated. Pregenerated vector graphics are given to the client just like

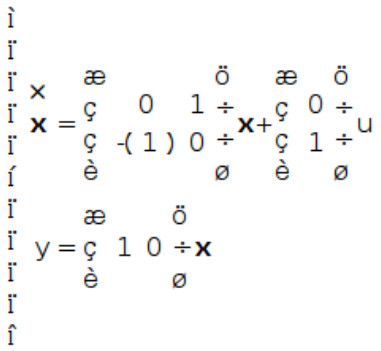

Figure 3.1: Failed rendering of mathematical content, due to missing fonts on the client side. The renderer should always check that the content will be perfectly rendered. The symbols missing here are the braces and brackets as well as a dot signifying the time derivative. Typically the biggest problems appear with multi-line symbols like integration signs and parenthesis. This example is produced by TtH with its default settings, when the user happens to access the material from an environment where the required fonts are not available. In this case the equations came from STACK 1.0, were converted for presentation with TtH, and the material was accessed with Mozilla Firefox on a Linux machine.

normal images but in vector graphics format like SVG. The generation of the vector image is done in the server. Client side generated vector graphics are normally created from some kind of higher level representation language like MathML [16] and require that the client understands that language. Naturally, the client can generate bitmap images from the representation language. The problem is that bitmap images require the same resolution for all users and will not work well with mobile devices, while vector based solutions require special[1] browser plug-ins or other software to work.

The most important feature of the GUI is that it must display all information perfectly, there is no room for "almost" perfect renderings of mathematical content (Fig. 3.1). Should there be any problems with rendering the system must instruct the student accordingly, for example to use another browser.

3.1.3 The controller

The controller executes the actions requested by the user through the UI and updates the model accordingly as in normal MVC applications. The special part in a CAA

[1]Both SVG and MathML work with Gecko-based browsers like Firefox, and certain other browsers have suitable plug-ins.

system's controller is that it usually controls some kind of an evaluation engine to evaluate grading algorithms.

The evaluation mechanism of grading algorithms is the centre of the controller. In a mathematical CAA system it normally contains an interface to a CAS or to some kind of a numerical software. Typically the CAS or the numerical software is an external program that will be interfaced by using a vendor provided interface or by a built-in software component. In some cases the CAA system contains its own implementation of CAS or a numerical system. But generally, using an implementation that has been designed specially for the class of exercises the CAA system is aimed at, tends to limit the possibilities of the system.

In systems that support parametrised exercises the controller also initialises the problems with suitable parameters. Even systems that only use static exercises may need to generate displayable output. Typically a CAS system inside the controller can do conversions from equations to mark-up languages or graphical representations fast enough, and there is no reason to implement such converter elsewhere in the system.

3.2 Exercise types

The main types are exercises with teacher-provided and student-provided answers. From the CAA systems point of view the first type is by far the simplest one to assess and to define. But from a pedagogical point of view the second type is more desirable. It is clear that the first type of exercises can always be graded by a CAA immediately. But in the second type the solution may well be ungradable for the system. From our point of view only those kinds of exercises that can be graded are of interest.

While these are the basic types of a exercise, one is by no means limited to just these, as they can be mixed to compose longer exercises with multiple solution phases.

3.2.1 Teacher-provided answer

This type includes all kinds of classification and multiple choice exercises. There are exactly two subtypes: the basic one is 1 of N where the student has to choose one and just one of the options, and M of N where the set of the chosen options forms the answer. Typical examples of these would be:

1 **of** N Choose the correct answer from these...

M **of** *N* Which of these statements are true...

The name teacher-provided comes from the fact that in multiple choice questions the answer is visible, the students task is just to choose the correct option. One could also consider a third type of an exercise, with more than one correct answer in the option set and the student can choose just one. But this exercise type would only be useful for checking, if the students actually calculate the answer by themselves or just choose the reverse approach to check out the options given, those calculating the answers could stop to think and maybe question the question, while those choosing the reverse approach would probably just choose the first correct option.

An important advantage of these question types is that the number of different inputs/answers for the CAA system is limited to a finite well defined set of answers, with no possibility of making syntax errors. It is also simple to build grading algorithms with these types of exercises. For 1 of N exercises just check if the answer is the chosen. If not, it could belong to a subset of options that are particularly bad and should be given special treatment (maybe even an punishment). For M of N exercises one normally works with the sizes of the intersection sets of the given answer and correct or the wrong options.

3.2.2 Student-provided answer

Exercise types with student-provided answer give a possibility to formulate almost any kind of an exercise as long as its answer can be written via the CAA system's UI. This means that the answer can be for example a number, an equation, a source code file or a drawing produced with tools provided by the UI, or even an essay. For a CAA system the input is normally limited by the syntax that can be parsed and evaluated by the system. This means that all but the last of the above examples could be dealt with by a CAA system. Even the last one, the ultimate free form input essay, could be searched for some specific content.

Text with syntax

Numbers and equations are examples of interest here, but exercises where the answer is given in this form also include programming exercises. Typically the answer is given in a single text field or, for mathematical context, there could be multiple fields to provide better presentation. For example a matrix could be presented as an array of fields. There are basically limitless possibilities for exercises. The limiting factor tends to be the grading algorithm. Typical mathematical examples are:

1. Give the derivative/integral/X transform of...

   These type of exercises are the simplest to check as they can be checked just by comparing the teacher's answer with the student's answer with a CAS. One simply subtracts the student answer from the teacher's answer and checks if the result simplifies to zero. The best part is that there is no need to even calculate the teacher's answer beforehand as the CAS can calculate it by itself.

2. Give an example of...

   Typically one asks for a function that has certain properties. For example, such that attains specific values at given points. Another example are matrices that show that the matrix multiplication is not commutative. Grading this type of exercises requires that the grading algorithm can be constructed from simple tests that check the properties and values of some mathematical objects. Defining this type of exercises tends to require more from the process of defining the grading algorithm and therefore differentiates the specially targeted and general CAA systems from each other.

3. Express Y using partial fractions/factor polynomial Z...

   While it is easy to check if an answer evaluates to same value as the teacher's answer, it is not easy to check whether the answer is in a specific form. This requires that we have simple ways of identifying and handling the terms of the answer. As this requires low level programming, and there normally exists multiple equivalent forms for an answer, building a bullet proof grading algorithm tends to be extremely hard. Luckily, tests for this are usually provided by the CAA system as ready made algorithms. Hence, assessing problems about e.g. partial fractions is not so complicated. Unfortunately these tests often rely on special use of the normal syntax and therefore may lead to a high number of false negatives.

Drawing

While free drawing may seem impossible for a computer to assess, it is still a possibility as it just means handling of a large set of pixels. There are plenty of algorithms for extracting information from raw images. But in the mathematical context one does not need free drawing, because lines and simple geometric shapes is generally enough.

With constrained drawing one needs only to work with the points and shapes, which can generally be described with a small set of parameters. Typically one

just gives shapes studied in some wrong configuration and the student is then asked to move them to the correct position. The system returns the parameters of the student's answer to the grading algorithm. Typically, the actual drawing is done with e.g. a Java applet which can be configured to allow only specific drawing operations and to output just the required parameters.

3.3 Tailored feedback

Typically a grading algorithm provides only basic feedback about an answer, but with tailored feedback one can focus on common errors with more precise feedback. Tailored feedback [6] is especially well suited for exercises where students are asked for an example of something. These types of questions could receive multiple answers where the simplest possible example is given while some possibilities newer get noticed. With tailored feedback one may give positive feedback to those students giving innovative or uncommon examples. For those giving the simplest solution some other examples could be shown.

Typically, tailored feedback is implemented within the grading algorithm by adding some extra tests that will trigger the extra feedback. Some systems might even provide basic tests that include some special feedback. For example, a system can contain a test specially meant to check if a student integrated a function correctly. The system may give special feedback if the student derivated the function instead of integrating it.

Tailored feedback is also used for giving the student feedback on specific features of the answer. For example in programming, it can be used to give specific warnings about some specific constructs in the answer [17]. In grading of programming exercises, tailored feedback is especially useful as an automatic notifier for simple errors even in the case that the exercise is not graded completely automatically.

3.4 Available web-based CAA-systems for mathematics

While there are multiple CAA-systems available for mathematics teaching, only few of them are currently in active development. Others have become obsolete due to low interest or just because of the old age. Next we compare the basic features of few of the most popular systems in the market.

3.4.1 Commercial systems

Commercial systems, in general, tend to provide more polished interfaces. On the other hand, one normally does not have access to the source code of the system. Hence one can never really know what happens inside it. Naturally, being a paying customer does bring some benefits in the form of technical support and services.

Typically the systems are provided on subscription basis. Hence the system may become useless the moment the licence or subscription period expires or shortly afterwards. This may be a big problem as the exercises written for a specific system do not necessarily work with other systems. If the license or subscription cannot be renewed one needs to rebuild the exercises in some other system. This might be a real risk as software companies might discontinue products or go bankrupt and will probably not open any license locks at that point.

Maple T.A.

Maple T.A. is one of the major systems available. Its pricing depends on the number of students and the level of support and maintenance services. One may buy Maple T.A. as software that needs to be installed on a server or as a service that Maplesoft runs on its own servers.

Being a Maplesoft product the system is built on Maple CAS and therefore has a rather solid foundation for handling the evaluation of the answers. As a slightly odd feature Maple T.A. has (at least up till version 2.5) also used its own evaluator (not Maple) to evaluate numerical answers in a slightly different syntax from the standard Maple. This evaluator is a remnant of an exercise system that Maplesoft acquired and which then become Maple T.A. This part of the system may cause some confusion for the user before he/she learns which evaluator is used in each situation. As a curiosity there also exists a way to write questions using LaTeX documents that can be converted to Maple T.A. exercises [18]. Basically, the documents map complicated HTML-form actions of the question editor to LaTeX macros that can be manipulated faster.

Maple T.A. is a complete exercise system with both class and user management. The newest version can even authenticate using LDAP. Maple T.A. can also be integrated to the Blackboard Learning System as a new question/quiz module.

There are plenty of ready made questions for Maple T.A. available from Maple Application Centre or from other sources. Naturally, the better question collections will have a price, but there are also free collections available.

An eContent project of the European Union (EU) named WebALT[2], has produced a large set of exercises for Maple T.A. The products of that project have been commercialised but what is interesting is that the project developed methods to produce multilingual exercises, nearly or fully, automatically to multiple EU languages using a mid-level language to generate translations of basic exercises.

MathXL

MathXL is somewhat different from other systems as it is purely a subscription based service that will not require any software installations to local servers. It does however have some serious limitations for the browsers that can be used with it. It uses its own specially built browser plug-in to present the content and the plug-in only works with Internet Explorer and Microsoft Windows.

MathXL is mainly targeted for use along with Pearson Educations[3] course books, and questions are made specially for them. It also allows creation of custom questions. Nevertheless it is obvious that MathXL is an add-on with goal of selling more course books and the corresponding question series. A student can even buy a course book in electronic form so that it can be used within the MathXL system.

While MathXL is a complete exercise system with courses and user management it has some limitations. For example the management of users may prove cumbersome if it is done through a browser. The biggest limitation is the browser plug-in requirement which basically limits the system for use inside a controlled classroom.

The main selling point for MathXL is the fact that there are ready made exercises covering entire course books. Naturally the exercises have been made especially for Pearson's books. One can only rent these exercises and never really own them.

3.4.2 Free systems

Free systems can be either open or closed source systems. Generally, most free exercise systems are open source systems as there is no reason to hide the source code. Concealing it would even be counterproductive because getting feedback or even bug-fixes from users could be harder.

All free systems are not truly free as some may require the use of a commercial CAS software, in which case the price of the system might be considerable but still much lower than with fully commercial exercise systems.

Like most open source projects the open source exercise systems do prefer using

[2] http://www.webalt.com/
[3] Addison Wesley and Prentice-Hall

open standards like XML and SOAP. The use of these standards makes it easier to modify the systems and to integrate them to other systems.

STACK

STACK (see Chapter 4) is a 100% free open source mathematical exercise system developed at the University of Birmingham under the leadership of Chris Sangwin. The reason for it being 100% free comes from the fact that the CAS it uses is one of the free CAS options available, called Maxima.

STACK is under constant development and its form is going to change from fully independent exercise system to an exercise-type plug-in for virtual learning environments (VLE) like Moodle. STACK versions prior to 2.0 are fully stand-alone systems with all the required features for running a course, but versions from version 2.0 STACK will require a VLE to work. From version 2.1 STACK will contain some new features presented in this thesis.

WeBWorK

Not to be confused with WebWork the web-application framework, WeBWorK [19] is a CAA system aimed for mathematics. It was originally developed in the University of Rochester but is now a group project with developers from many universities and high schools around the world. It is licenced under the Artistic License originally used for Perl and is basically open source software, although the Free Software Foundation may think otherwise [4].

WeBWorK is an complete exercise system with user and class management. It uses Problem Generating (PG) language for defining its exercises. The PG language consists basically of Perl macros and a carefully selected subset of Perl where the parts that could cause damage like file access have been disabled. As the exercises are defined in a highly expressive language like Perl, one can do pretty much anything with the system, but some users may think that too much freedom makes things too complex. Because WeBWorK does not use any kind of a CAS, it does the evaluation of symbolic formulae in Perl. Consequently, some things that are easy on systems built on top of some CAS, could prove very complicated to do in WeBWorK, but on the other hand some things are much easier, because WeBWorK acts on a higher level. It has some interesting features like unit conversions that could be useful for physics exercises.

[4] http://www.gnu.org/philosophy/license-list.html referenced 16.3.2008.

	MathXL	Maple T.A. 3.0	WeBWorK	STACK 1.1	STACK 2.0
CAS	Mathematica	Maple	Own (Perl-Based)	Maxima	Maxima
Database	?	Postgre (SQL)	GDBM (not SQL)	MySQL	MySQL[1]
Language	?	Java (JSP, J2EE)	Perl	PHP+maclisp	PHP+maclisp
Platform	?	Tomcat servlet container	Apache HTTP server	PHP	PHP + Moodle
Output rendering	Custom browser plug-in	Images / MathML	TtH / Images / LaTeX2HTML	TtH[2]	TtH[3]

1 Version 2.1 includes ADOdb abstraction layer to support most standard SQL databases.

2 Our modified version renders as images.

3 Moodle filters are being investigated for future versions.

Table 3.1: Basic technology comparison of some mathematical CAA systems.

As a speciality, WeBWorK has a fine system for E-mail messages. It can easily generate personalised messages to students based on templates written by the teacher. These personalised messages can contain student specific personal information as well as scoring information.

3.4.3 Comparison

Comparing these systems with each may be somewhat biased as each of them has some kind of a special feature that none of the others have. Even spotting this feature may be hard when we inspect features of interest. Nevertheless, the following tables compare MathXL, Maple T.A., WeBWorK, and STACK versions 1.1 and 2.0 as well as the modified version of STACK.

	MathXL	Maple T.A. 3.0	WeBWorK	STACK 1.1	STACK 2.0
Question pools[1]	yes	yes	?[2]	no	yes (Moodle)
Prerequisite control[3]	yes	no	no	no	no
E-mail[4]	yes	yes	yes	no	yes (Moodle)
Raw text answers	yes	yes	yes	no[5]	no[5]
Localisation support[6]	English	English	English	English, Spanish, Finnish[7], French, Dutch	English
Unicode support[8]	?	no[9]	yes	yes	yes

1 Whole questions are picked at random from a pool of questions i.e. quiz may have different questions for different students not just the same questions with different parameters.

2 If not directly available there is always the option to code it manually.

3 Force specific questions to be done before accessing others.

4 Does the system provide means to send messages to student's E-mail.

5 Inputting raw text as answers is not possible in STACK as all input goes through the CAS.

6 Is the user interface available in other languages?

7 Finnish translation is underway at TKK.

8 Does the system support Unicode character encoding, for writing exercises that have localised content?

9 There is an UTF-8 version coming, we have tested a development version of 3.01 that does support UTF-8.

Table 3.2: Feature comparison of some mathematical CAA systems.

	MathXL	Maple T.A. 3.0	WeBWorK	STACK 1.1	STACK 2.0
Parametric questions	yes	yes	yes	yes	yes
Symbolic answers	yes	yes	yes	yes	yes
Numeric answers	yes	yes	yes	yes	yes
Multiple input fields	yes	yes	yes	no[1]	yes
Inline input fields	yes	yes	yes	no	yes
Geometric / drawing exercises	yes[2]	no	no	no	no
Matrix input fields	yes	yes	yes	no[1]	?[3]
Multiple choice fields	yes	yes	yes	no[1]	?[3]
Essay questions	yes	yes	yes	no	no
Grading algorithm structure	?	free	free (Perl)	tree	forest
Parameter selection[4]	implicit	implicit	explicit[5]	explicit	explicit
Feedback[6]	static	static	dynamic	dynamic	dynamic
Penalties and points	max points - penalties from tries	points from a function - penalties from tries	free (Perl)	each node of the tree may affect both grade and the penalty.	

1 Included in our modified version.

2 Control point manipulation for predefined objects.

3 Implementation ready, should appear in version 2.1.

4 How the parameters of a parametric exercise are selected, basically, one either defines exactly which parameters are suitable or defines the conditions that the parameters need to meet.

5 Can be programmed anyway one wishes, including iterative testing with implicit conditions.

6 Does the feedback vary depending on the answer i.e. can one give specific feedback on specific problems or is the feedback just static correct/incorrect message.

Table 3.3: Question authoring comparison of some mathematical CAA systems.

Chapter 4

STACK

The CAA system chosen for experimental courses in Helsinki University of Technology is STACK. The choice was rather simple because STACK is a free open source software and therefore gives the possibility to set up systems without constraints caused by software costs. At the same time it is an open source system that can be modified to suit emerging needs. Some other options were tested extensively, but they were found unsuitable at the time the choice was made. The ease of use and logically consistent structure of the system were also good reasons for choosing STACK.

4.1 History of STACK

STACK was released near the end of 2004, and by the summer of 2005 it had reached its form and most of its current functionality. After that its appearance has been polished and its performance been developed. STACK's relatively fast start might be explained by the fact that its fundamental ideas are based on AIM and that the project leader for STACK, Chris Sangwin, had extensive experience on AIM. Nevertheless, AIM and STACK are completely different from the inside. The code for STACK has been written in different programming language than AIM's, the CAS used is different as well as the underlying server. The only thing common between the systems is the use of TtH as output formatter.

The version of STACK used at Helsinki University of Technology comes from the STACK 1.x branch of STACK development, being the version 1.1. The version 1.0 was released in March 2005. Version 2.0 was to be released by the end of 2007, but it now seems that version 2.1 will take its place around the summer of 2008.

4.2 Technologies used in STACK

STACK is a XAMP application, meaning that it works on a open source web platform consisting of the Apache web server, MySQL database server and PHP preprocessor. It runs on most operating systems supporting those and few other applications. Basically this means Microsoft Windows, Linux and other POSIX platforms.

STACK uses TtH to generate CSS/HTML code from internal LaTeX-representation to display mathematical content in browsers. Unfortunately TtH is not so well supported by all browsers. Where it works it is fast and on the server side it is really fast, and the generated data takes little space to store. MathML presentation could eventually become an better alternative for TtH in STACK but the problem is the lack of browser support.

In the core of STACK there is an open source CAS named Maxima, a GPL licensed program that has been developed from a 1982 version of the original Macsyma. Maxima suits rather well for the needs of CAA system, because it is a general purpose CAS that allows creation of user-written functions needed for specialised checking. Although Maxima does not provide as attractive a graphical user interface as certain other modern systems do, it works perfectly inside the core of a CAA system. Maxima can even be used for plotting, so it contains all things needed by the core of a CAA system.

4.3 STACK in parts

Here we go through the basic parts of a CAA from the point of view of the MVC pattern as defined in Chapter 3.

4.3.1 The model

STACK's data model is rather flexible and can handle most situations. It does have one slight flaw in the definition of user rights, namely that there is just one administrative user and one can not give any of its rights to other users. STACK does provide ways to define groups of students that have access to groups of exercises, and the rights model allows making exercises public and accessible without a user account of your own.

The model stores all answers and the related scores as well as input data that was given just for validation. Everything is accessible to the administrator. The data is directly linked to the students and can therefore be traced to them.

The model stores all exercises as templates that may generate different exercises

based on a random seed number given to the template. The exercises can be used in multiple exercise series around the system, and even the series can be used in multiple courses.

The grading algorithms are defined as a decision tree. Every node of the tree contains an answer test function, which takes up to two variables defined freely (using a very expressive language) and returns a boolean value to decide to which branch of the tree the algorithm continues. All visited nodes of the tree may modify the final score based on their boolean values. Typically the test function tests the algebraic equality of the student's and the teacher's answers. Each node of the tree can also contain feedback for the student. The model scales well from simple single test algorithms to massive trees handling the identification of dozens of classes of answers while at the same time being simple to use.

4.3.2 The view

STACK's user interface displays students a hierarchy of possible exercises as follows:

1. Subjects are the highest level of the hierarchy. They might as well be considered as whole courses. One may define a list of students, who can access the subject contents.

2. Quizzes are placed under subjects. They can be considered as small exercises. One may define the time a quiz closes[1] as well as various rules related to the grading of the quiz.

3. Questions are the exercises and they are placed inside quizzes.

Inside quizzes STACK displays questions belonging to the quiz either one by one on separate pages or all in the same page, depending on the preferences of the student. All STACK's questions are displayed by printing the question with whatever mathematical content through TtH on the page over one text field, to which the student should give his/her answer. After writing the answer to the field the student may choose either to validate or mark the answer. Validation reads the answer and checks the syntax as well as shows the student the answer rendered in more readable mathematical form. Validation is an optional action but it is highly recommended, because choosing the mark action directly might cost points, if the answer is interpreted differently than intended due to syntax errors. Depending on

[1] Opening time cannot be defined in advance in version 1.x, Moodle allows this in 2.x. One can use `cron`-jobs and database manipulation if one needs to set the opening time at an inconvenient time.

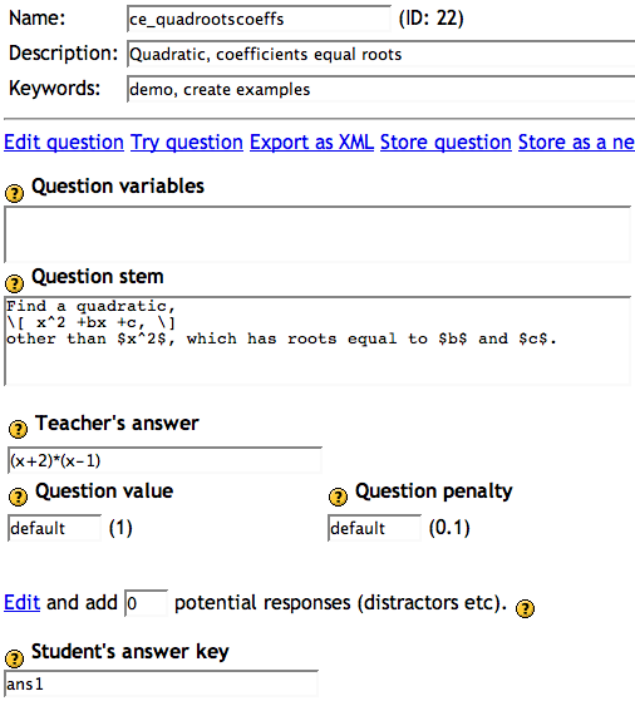

Figure 4.1: First one defines any question variables one might need e.g. if one wishes to randomise some parameters. Then one gives the question stem in which one may use those variables if one wishes. After which one should give the correct answer and tell to which variable the student's answer should be stored.

the grading settings one may try to submit a new answer if the original one was wrong. In some cases the system even displays the correct answer and solution when asked. Typically, the solution will be displayed after the quizzes deadline has been reached.

For a teacher STACK provides tools to write questions as well as analyse answers to old questions. Writing new questions is rather simple but also allows extremely complicated questions to be constructed. In the simplest case one can create a question by filling just two fields in a form e.g. "Derivate x", and the teacher's answer "1" (Fig. 8.3). By default STACK will compare the student's answer algebraically to the teacher's answer. More complex questions require some defining of CAS variables and more tests for the answer, but the logic of defining questions is the same. Even though the forms (Figs. 4.1–4.3) for defining a question may be large they are rather simple.

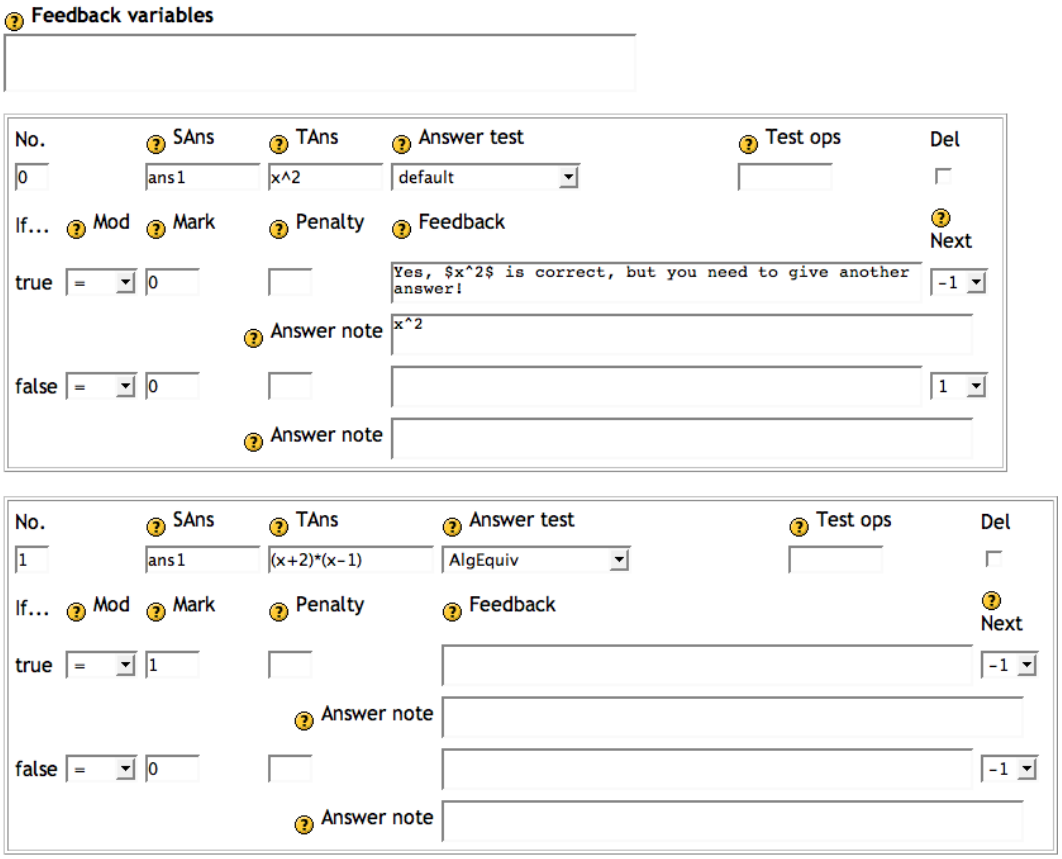

Figure 4.2: Second part is the definition of the grading algorithm. First on may wand to calculate some values from the students answer. Then comes the grading algorithms tree, that is defined node by node, each node defining the test to be done and any feedback given as well as the connected nodes.

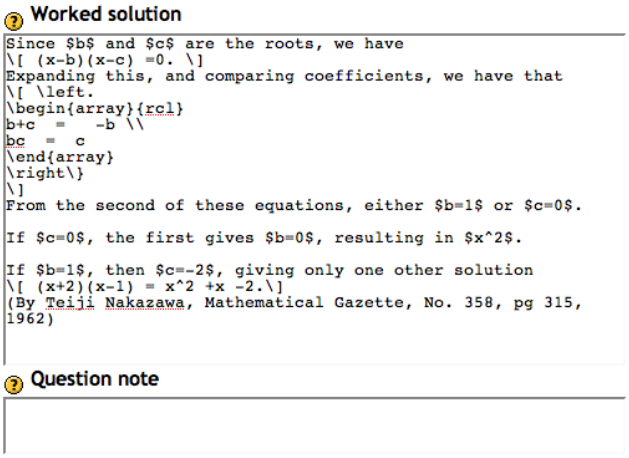

Figure 4.3: Lastly one may add a solution and even some notes to be stored for analysis.

4.3.3 The controller

The controller part of STACK has been constructed around Maxima, which is used for evaluating tests needed for grading as well as generating LaTeX-code for presentation. Previously STACK did create a new Maxima process every time something needed to be evaluated, but now there are versions with server implementations of Maxima with better efficiency. Maxima is rather fast to start, but it is still expensive to start new processes, hence keeping Maxima running as a server process provides better performance.

Short history of Maxima

Maxima is a CAS with a long history that can be traced back to 1968. Then it was called Macsyma, and it was one of the first symbolic computation systems. Many of its ideas have later been adopted to new systems like Mathematica from Wolfram Research and Maple from Maplesoft.

The first part of Maxima's development took place at The Massachusetts Institute of Technology as part of The Project MAC during 1968–82. Back then Macsyma was written in Maclisp, a dialect of Lisp designed at the Project MAC. As a side-note the first Lisp based Emacs was also implemented in Maclisp.

In 1982 the development diverged as the code was licensed out to Symbolics and the United States Department of Energy (DOE). Symbolics started to develop their own version behind closed doors, while the original version's development slowed

because of disputes related to licensing. The commercial version of Macsyma was rather successful. For example in 1995 PC Magazine said "Macsyma is now the leader in the mathematical program marketplace." in a review [20].

In 1998 the public life of Maxima began as a branch of the original Macsyma back from 1982 was released under the GNU General Public License. This version was in the state it was left in 1982 and therefore required plenty of work to bring up to date. The fact that the Symbolics version of Macsyma had been developed using about 50 man-years since 1982 makes it nearly impossible to reach the same level. But still Maxima has been developing at a good pace. A temporary setback was when the maintainer of the code William Schelter passed away 2001. William Schelter adapted Maxima to Common Lisp and enhanced the system in many ways, and thanks to his efforts the Maxima we know is now available for us.

Chapter 5

Statistics from course KP3-I

5.1 Introduction

The basic course in mathematics KP3-I is a one period course introducing certain new concepts to second year students. The students come from either the Department of Forest Products Technology or from the Department of Chemical Technology. The main topics of the course include:

- basics of complex analysis,
- integral transformations (i.e. the Laplace transformation),
- and the Fourier series with applications to differential equations.

When the course was lectured in autumn 2006 STACK software was tested as a way for the students to submit a part of their voluntary homework exercises in the web. Traditionally the students were given bonus points for those exercises that the students were prepared to present at the exercise session. These points could basically raise grade by one grade[1]. With STACK we added exercises with immediate automatic assessment and another possibility to collect some more points.

The STACK software was used on this course as just another way to submit the solutions. The students could use the so-called service password to login to STACK any time they wanted from anywhere. The questions were given on paper like before. The system was tuned so that it only graded the first attempt. The students could not retry if the answer was wrong. But the system still allowed the students to continue inputting new answers and checked them to tell the students if they were correct or not.

[1]The grades given are integers 0-5.

5.1.1 Size and length of the course

While the course is not one of the largest basic mathematics courses given at Helsinki University of Technology it still typically has about 200 students. In autumn 2006 207 students enrolled in some way to the course. Some of them did not enrol to any of the exercise groups and came just for the exam.

The course began with a lecture at 8.9.2006 and the end lecture was held at 20.10. There were three lectures a week and the lectures were one and half hours long. The traditional exercise sessions were arranged each week for five exercise groups that had between 36-52 enrolled students. In reality the number of students present was much lower. There were two guided sessions for these groups each week, the second session was a demonstration session where the assistant teaching the group went through some examples. The exercises had the following timetable:

Round	Theme	STACK deadline
1.	Basic complex number representation and changes between forms.	Tue Sep 19 23:59:59
2.	Cauchy-Riemann equations, splitting equations to real and imaginary parts and some derivatives.	Sun Sep 24 23:59:59
3.	Analytic functions, partial fractions and integrals in the complex-plane.	Sun Oct 1 23:59:59
4.	Möbius transformations.	Sun Oct 8 23:59:59
5.	Fourier series.	Sun Oct 15 23:59:59
6.	Laplace transformation.	Sun Oct 22 23:59:59

5.1.2 Modifications to STACK

The system was slightly modified for use on the course. Among some rather cosmetic modifications were the following:

- Replacement of the authentication system with our own version that connected to the university's SSO-system. This modification also meant removal of some user information related options from the STACK's user interface.
- Image based rendering code for all generated content was installed so that the equal representation of the exercises could be ensured. This was done

primarily because the TtH software that STACK uses is aimed for Internet Explorer and does not work very well with the browsers of our target audience.

- As a part of the authentication system a logging system was also installed so that in case of problems we could track all of the actions of the users. The logging system was built to collect normally unlogged data like the POST parameters and for certain pages the whole session state.

- The session data management code was also replaced to stop leakage of session data in cases where the user used two different STACK systems at the same time.

5.2 Data collected

The systems logging capabilities were used to collect various different statistics about its use and possible problems. Particular attention was given to possible input errors.

5.2.1 Basic activity data

One of the more obvious statistics collected was about the usage of the system, that is when the students used the system. Unsurprisingly the results (Fig. 5.1) show that, when given a chance, students submit their exercises at the last possible moment. On the other hand (Fig. 5.2), students seem to submit exercises at any time of the day, even when listening other lectures.

5.2.2 Answer lengths correlation with syntax errors

Because some of the exercises on the course required rather long answers. It was interesting to see how long answers can one expect the students to actually input using this system. Table 5.1 shows that when the answers length rises to over twenty characters, as it often does, the likelihood that a student makes an syntax error rises to over 10%. This means that one should try to minimise the length of the answer.

5.2.3 Types of invalid input

The students of the course produced 257 different invalid inputs. By analysing these one can find at least the typical input errors listed in the table 5.2. The most obvious reason for syntax errors is that the students do not understand how the answer is parsed in the system.

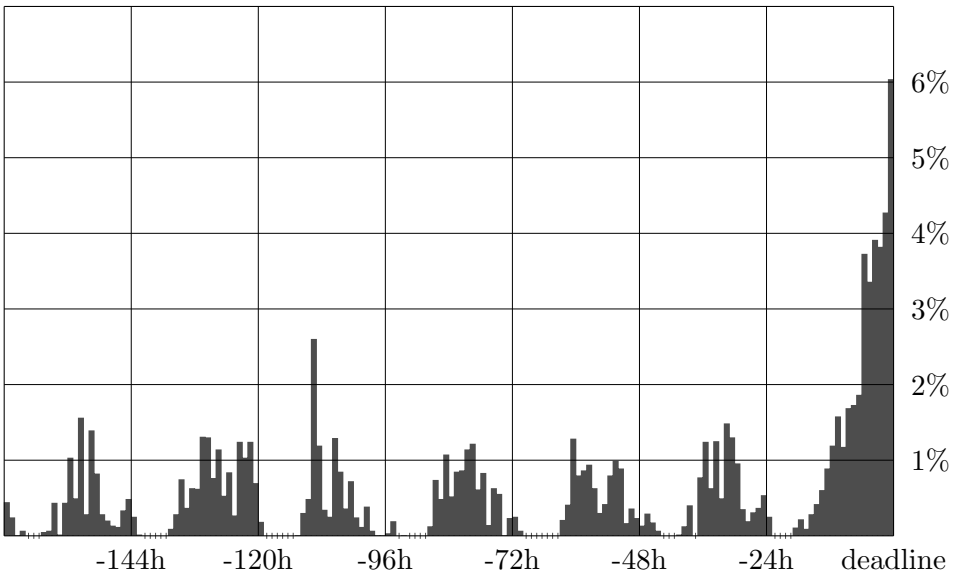

Figure 5.1: Activity in the system as time nears to the deadline. It is not too surprising to see that most activity concentrates to just before the deadline, here the activity of the last hour counts as 6% of the total activity. This histogram consists of hour wide bins and it has been scaled so that the heights of the bars show the percentage of the total activity falling to that particular hour.

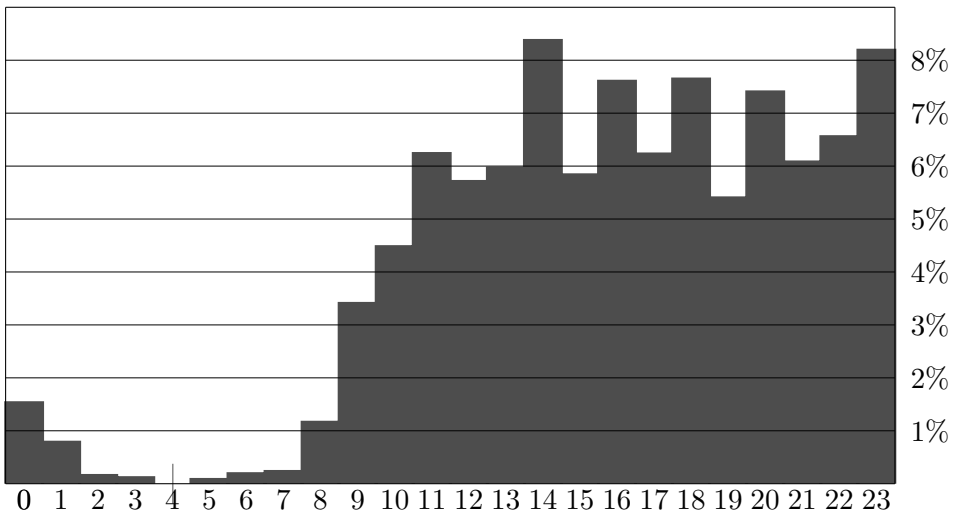

Figure 5.2: Activity in the system by hour of the day. Interestingly no one does anything at 4 o'clock in the morning.

Length	N	Valid	Invalid	%
1-4	1793	1763	30	1.7
5-9	630	594	36	5.7
10-14	503	487	16	3.2
15-19	846	816	30	3.5
20-30	672	582	90	13
30-50	395	325	70	17
60-100	97	59	38	39

Table 5.1: The likelihood that the students input is invalid rises as the length of the input grows. This is naturally understandable, but due to the low sample size of the longest inputs their high error rate is uncertain.

Based on the data the students should be told at least the following examples so that they understand why the input can not be understood:

- For the system `xy` does not mean the multiplication of `x` and `y` it is the name of the variable `xy`. One must always use the multiplication symbol/operator `*` to tell the system to multiply.
- `x(x^2+1)`≠`x^3+x` as without the multiplication sign between `x` and the parenthesis the system thinks that you are calling a function named `x` just like you would call the sin function `sin(x)`.
- `[(x+y)*sqrt(x)+1]/y` is a vector because the system uses brackets to define vectors. Also curly braces are used to define lists so you should remember to only use basic parenthesis to group terms.

5.2.4 Number of tries

Because the exercises could be done again after the first graded attempt it was interesting to see how many times students do them. It was expected that those who did not get the correct answer on the first try would try again but unexpectedly according to the data (Table 5.3) even those who did answer correctly tried again. This result can be explained with exercises that asked for an example of something, as those exercises naturally make the student think about other choices.

Category	N	%	Description
Parenthesis	76	30	
unbalanced	72	29	Not closed or mismatched.
wrong type	4	1.5	Use of brackets and curly braces when they have other meanings. `x^2/[(x+1)^2+2]`.
Equals sign	76	30	Returning equations or definitions when only the value was asked for. `y=e^(-t)`.
Multiplication	95	36	Lack of multiplication sign
xy	60	23	Separation of variables.
$x(y+x)$	65	25	Mixed with function syntax.
Wrong symbols	9	3.5	Typically π written as `pii`. In Finnish language one spells it like that.
Syntax errors	22	8.5	
arguments	18	7	Separation of the arguments of a function, `cos y`, `sinhi`.
mixed syntax	5	1.9	Contextual problems typically arising from powers of trigonometric functions like `sin^2(x)`.

Table 5.2: The most common input errors found in the students syntactically invalid answers. And the percentages of the invalid input strings containing them.

In any case it is promising that 43% of those who answered incorrectly tried again. And this number could probably be raised just by motivating them to try again, just giving bit less points would probably increase the number to nearly 100% and advertising the possibility to try again without any penalties or risks would probably also increase the numbers.

Responses	First correct	First incorrect	Total
1	865 (77.0%)	284 (56.9%)	1149 (70.8%)
2	211 (18.8%)	124 (24.8%)	335 (20.7%)
3	31 (2.8%)	43 (8.6%)	74 (4.6%)
4	12 (1.1%)	23 (4.6%)	35 (2.2%)
5	3 (0.3%)	13 (2.6%)	16 (1.0%)
6	1 (0.1%)	5 (1.0%)	6 (0.4%)
7	0 (0.0%)	2 (0.4%)	2 (0.1%)
8	0 (0.0%)	2 (0.4%)	2 (0.1%)
≥9	0 (0.0%)	3 (0.6%)	3 (0.2%)
Totals	1123 (100%)	499 (100%)	1622 (100%)

Table 5.3: The number of different returned values tells us that some students do try to give the correct answer even if the first attempt was wrong and when no points can be gained from subsequent attempts. Surprisingly 23% try again even when they had the right answer at the first attempt, this might be because some of the exercises were of the type "Give an example of...". The most important thing is that 43% of those who have made a mistake with the first answer will try again.

5.2.5 Higher interest

Compared to the traditional exercises the STACK exercises were by far more popular (Fig. 5.3). Although we only have five real data points it would seem that the number of students doing the STACK exercises is considerably higher. It's also interesting to note that the number of students seems to vary less from week to week for STACK exercises, although this cannot be confirmed from this small sample size.

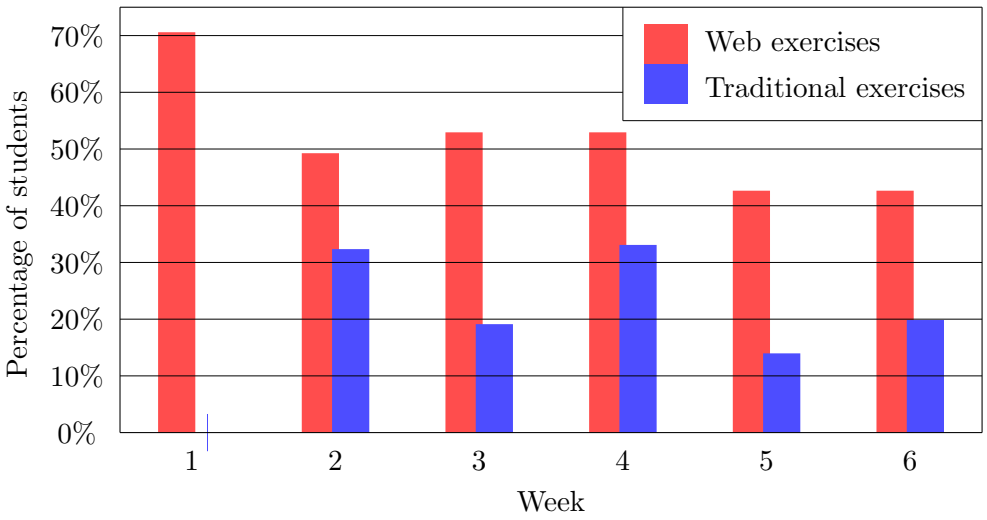

Figure 5.3: The number of students returning answers to traditional exercises was much lower than to STACK exercises. and while the interest dwindled as the weeks passed it was always higher than the interest for traditional exercises. There were no traditional exercises for the first week. The percentages are of the total enrolled students enrolled to the corresponding exercises.

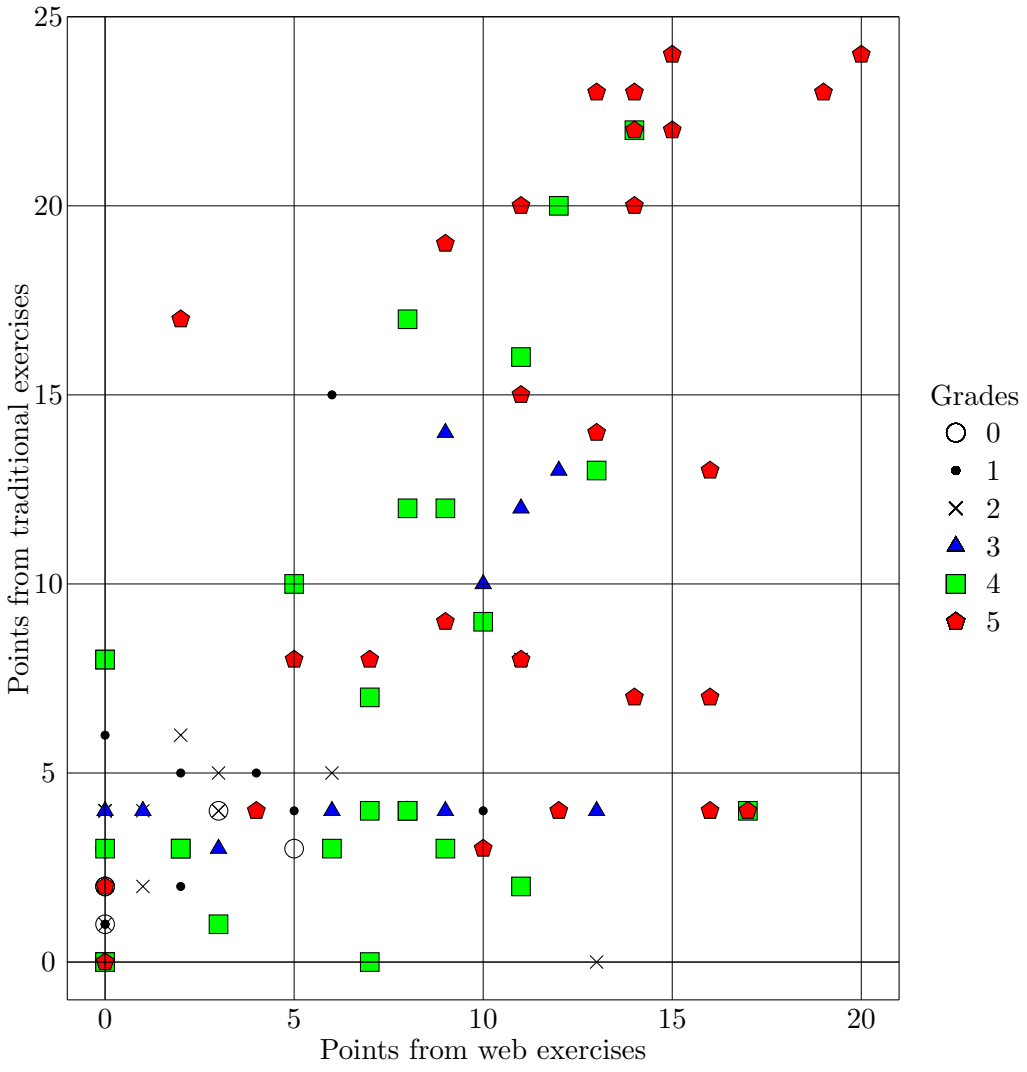

Figure 5.4: There is an obvious correlation between the points gained from exercises and the course grades. But what is interesting is that students seem to favour the web exercises. The traditional exercises had one easy round with a feedback question that basically meant that almost everyone gained at least four points from the traditional exercises, which explains the base level. This data includes only the 125 students that took part on the first exam of the course and might therefore be biased as the better students tend to take the first exam.

Chapter 6

Integration to supporting systems

6.1 Authentication system

STACK contains its own authentication and user management system, but for larger scale use it needs to be replaced with a system that provides more functionality and certain features not directly available in STACK. For example, defining new user accounts just for use in STACK with new passwords is by far too inconvenient for students and administrators to be used on a large course, not to mention the added work for handling these accounts for multiple courses.

There are four distinct problems that should be solved:

1 Too many passwords. As long as possible one should avoid creating new accounts that require new passwords.

2 Defining the user sets. Typically we already know the students enrolled to our courses, and there should be an easy way to transfer this list to STACK and use it to limit user access.

3 Handling the accounts and user sets for multiple STACK installations.

4 Ensuring that multiple STACK installations can run on the same server and at the same time be operated by a user without risk of leaking session-data.

6.1.1 Single Sign-On technology

Single Sing-On (SSO) is defined as a technique where a user authenticates to a trusted party. The trusted party then informs the party that requested for the

authentication that the user is who he/she claims to be. SSO-services can also give the information required to decide what kind of a user account the user should have. We have implemented a SSO system as the solution for Problem 1. in the above list. In our case, Helsinki University of Technology already provides a SSO-service which can be used to identify users. This service provides certain key pieces of information about an authenticated user to the service requesting the authentication. For our purposes we require the e-mail address, the student number and the account name of the user, which are easy to obtain. In the future we could also ask for the SSO-service if the user has enrolled to the course.

Our trusted party uses Shibboleth as the SSO implementation and therefore we also use in at our side. Because Shibboleth is based on OpenSAML communication interface, one could use some other system as well. Shibboleth is an open source project belonging to the Internet2 community's Middleware Architecture Committee for Education (MACE).

Our solution to Problem 1 is as follows:

- Instead of using STACK's own login page we use the SSO-service to authenticate users.
- The information provided by the SSO-service is used either to pick one of the existing user accounts in the STACK system or to create a new one. New accounts are only created if user's account name or student number can be found from a white-list of users that are allowed in the system.
- When the account has been found or created we just set it as a logged in account and let STACK run as normal. If the user's account name is found from a white-list of users with administrative privileges, we let the user choose if he/she wants to login as an administrator.
- For logout action we add additional listeners to relay the logout action upwards to the SSO-service.

The solution simplifies both the administrator's and the user's work. The users do not need to memorise new passwords for STACK. The administrators do not even need to create the user accounts. They just list the users, and the system creates the accounts when they login for the first time. This solution also partially deals with Problems 2 and 3. For Problem 2 it only handles access control to the entire STACK installation and does not handle the more fine-grained access control inside the system. For Problem 3, it works just fine as it can act as the gatekeeper for multiple STACK installations and handle different white-lists for all.

The white-lists will be initially defined as files listing the users that have access to the service. In the future, we intend to ask the SSO-service to take care of the course registration information and use that as a way to control access for students. But before the SSO-service is able to provide this information we must implement the system so that it can be easily configured to receive the information from different sources.

Single Sign-On process

There exist multiple ways to provide SSO. Shibboleth uses the so called federated identity-based authentication and authorisation infrastructure built on Security Assertion Markup Language (SAML) messages. SAML basically defines a protocol for exchanging authentication and authorisation data in an encrypted XML-form. The process for authenticating a user using some user agent (UA) (e.g. a web browser) for access to some service providers (SP) service (e.g. our version of STACK) by using some identity provider (IdP) is displayed in Fig. 6.1. The phases of the process are:

1. The user sends a request to the SP for a secure resource that requires authentication.

2. If the user is already authenticated the SP provides the requested resource. Otherwise the SP responds with one of the following:

   (A) a login form that submits to the IdP and contains a Base64[1] [21] encoded SAML message. The message basically contains just the name of the SP and the URL of the requested resource for re-routing, but it might contain more information.

   (B) a HTTP (302) redirect to a login page provided by the identity provider. The redirect contains a GET parameter with the same Base64 encoded SAML message as in case (A).

   In both cases the IdP will receive the information needed to route the user back to the requested resource after the password and user-name received from the user have been deemed valid.

[1] Base64 is a standard way of encoding binary data into a set of common characters that can be reliably transmitted among normal character data. The 64 characters used in the coding can be found from most commonly used character sets. Base64 is 75% efficient it stores six bits to each eight bit character.

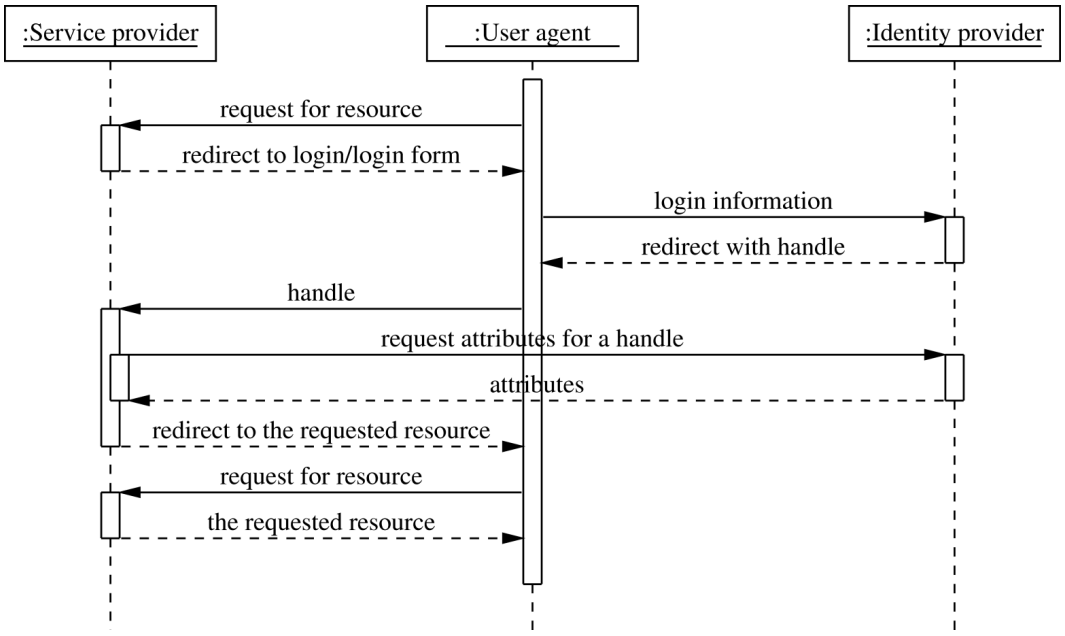

Figure 6.1: Single Sing-On process for authentication of a user. The situation here displays only one identity provider.

3. When the authentication has succeeded the IdP responds with a redirect to the SP. This redirect contains a handle that the SP can use to request more information about the user from the IdP.

4. The SP sends a request to the IdP for more information on the user. The basic user-name is already known, but e.g. the e-mail address or some kind of an employee number or student-Id could be demanded.

5. The IdP responds with the information, if the SP has been given the right to access such information.

6. Now that the SP knows the user it sends the user a redirect to the original requested resource or to some other place like the front-page.

7. The user sends the request for the resource defined in the previous redirect and we the process starts again from 1, but now considering that the user has already been authenticated.

6.1.2 Session management and tracking

Web applications typically store some information about the state of the system and about the user progress to the server. This information is called the session data. Handling the session data should be done with extreme care, since it represents the state of the web application, and if the state becomes corrupted then serious problems may arise.

One may regard a web application as a state machine which has its state stored in the session data. The user requests trigger state transitions. We can track the whole session as seen by the server simply by storing the state in some key points together with all events that trigger state transitions. Previously one has typically tracked only a small part of the user requests, that is the simple parameters send by the GET-method [22], but for complete tracking we also need the more complicated POST parameters and the state of the system at some specific points.

Because STACK is a PHP-application we can easily access all the session data it generates, and even replace the standard storage method with our own. We must do this as we want to run multiple STACK installations on a single server. STACK's session handling has been implemented such that it uses the default PHP session storage system directly and stores data to it without any kind of installation specific naming. This means that if one would use multiple STACK installations on the same server, the installations would write their session data at the same place quickly causing session leakage and, finally, reaching a point where session data represents an invalid state.

As a solution to this and to Problem 4 we replace the default PHP session handling with our own implementation. Our version will store each STACK installations session data separately and, at the same time, collects tracking data. The system will store the data to a database for easier navigation and gives an option to define, on page by page basis, how much tracking data is collected. The system will collect both GET and POST data, snapshots of the session data for some pages, and even some HTTP headers.

Example of session leakage

Consider a server running two installations of STACK, each with different administrative passwords. When one logs in to one of them as an administrator, STACK will write to the session data that the user has logged in as an administrator. Now assume that one does not log out and goes to the other STACK. That reads from the session data that the user is logged in as an administrator and acts as the user were

an administrator. Luckily a student cannot become an administrator but he/she could take the role of a different (almost random) student and see and do something that he/she most definitely should not be able to.

This is a major problem because the main reason why one would like to have multiple STACK installations is that there is only one administrative user account in a STACK system. You therefore need separate systems with separate passwords if you wish to limit the possible damage that one could do with such an account. It is also much easier to handle multiple courses on the same server with multiple independent systems instead of using a single system.

6.2 Other interfaces

Some data inside STACK should be accessible to certain tasks like grading. For this reason we will implement interfaces that take input in the form of POST messages. These interfaces are meant both for human and machine use and output their data in XML form, which is displayed to humans trough XSLT-stylesheets.

6.2.1 Grading information

Grading information is typically needed for use in spreadsheets or other tools and should therefore be available in easily copied form. This interface gives access to both student and exercise specific information.

The interface has been implemented as part of the administrators' tools so that he/she can request a table to be outputted with specific students listed with scores from specific exercises. To make this interface directly usable the table can contain columns that can have their values defined by Reverse Polish notation (RPN) expressions. Thus one can also use it to calculate total scores without the need of a separate spreadsheet program. The interface has also been designed such that it can be used to generate public score pages, where the scores can be displayed in real-time to the students.

6.2.2 Exercise information

For the grading interface we need to know the names of the exercises we are interested in. For other purposes we might want to know which exercises are currently open and how many students have answered to them. While this information can be found directly from inside the STACK's administrators' user interface, automated systems like timetables on websites need to have access to it directly.

For this purpose we simply add an interface that lists all objects in the system and the current status of those objects. The list will give ID-numbers to each object for use in other interfaces. Objects include questions, quizzes, and subjects as well as active user accounts.

Chapter 7

Modification of the user interface

While studying data from a test course where STACK was used (Chapter 5) we found some areas where improvements are necessary, mainly in the input system. Due to STACK's structure, most modifications can be done simply by adding some extra processing into some phases of the normal operation cycle.

7.1 Input modifications

Modifications to the input system of STACK are rather straightforward and do not require any modifications to the core of STACK. Basically these input modifications map modified input types to the traditional input model and back. The modifications are implemented at the highest levels of the system. Some modifications do however need access to certain mid-level functions like validation and answer tests, while others rely on modification of the instantiated question to guide STACK to do something more.

The additional parameters required by these modifications are stored in the question prototype and are interpreted when the question is instantiated. Typically this means adding new variables to the question instance and placing markers for various rendering modifications.

7.1.1 Typed input

Normally an exercise only receives answers of just one type i.e. exercises normally accept only scalar value expressions or matrices, not both. Typically one may con-

sider an answer of the wrong type to be some kind of an input error that should be dealt with differently. We define the following types and handlers:

`free` This type accepts anything and gives no warnings unless defined elsewhere. By default everything is considered to be of this type.

`scalar` This type accepts anything that looks like a scalar, i.e. the input must not contain any notation used in inputting vector, matrix or list types. Warning is given if such notation is found.

`vector` This type accepts vectors and will give a warning if the input does not contain necessary brackets or if there are unbalanced or extra brackets.

`matrix` This type accepts matrix's and will warn if the input does not contain necessary brackets or if there are unbalanced or extra brackets. It accepts only 2D-matrix-es.

`list` This type accepts lists and will warn if the input does not contain necessary braces or if there are unbalanced or extra braces. It accepts any kind of a list, the list may contain lists and anything else.

To define and check the type we provide two means:

A Input variable typing, when defining the name of the variable that stores the student's answer. The type may be defined at the same time. It is selected from a drop down list and gets encoded to a variable of the question prototype.

B Answer tests that can be used for any data are available. Unlike the previous option these tests are not carried out at the validation phase and are therefore intended for cases where one can actually input multiple different types and the answer test depends on the type of the input.

Unfortunately, this idea was newer implemented, because the most problematic cases could generally be handled with the other modifications made to the system. Basically, the implementation of typed input was not a priority since matrix inputs were implemented as 2D inputs (7.1.3)

7.1.2 Multiple input

The basic example on STACK's capabilities is the question where the student is asked to provide two matrices **A** and **B**, such that the matrix multiplication is not

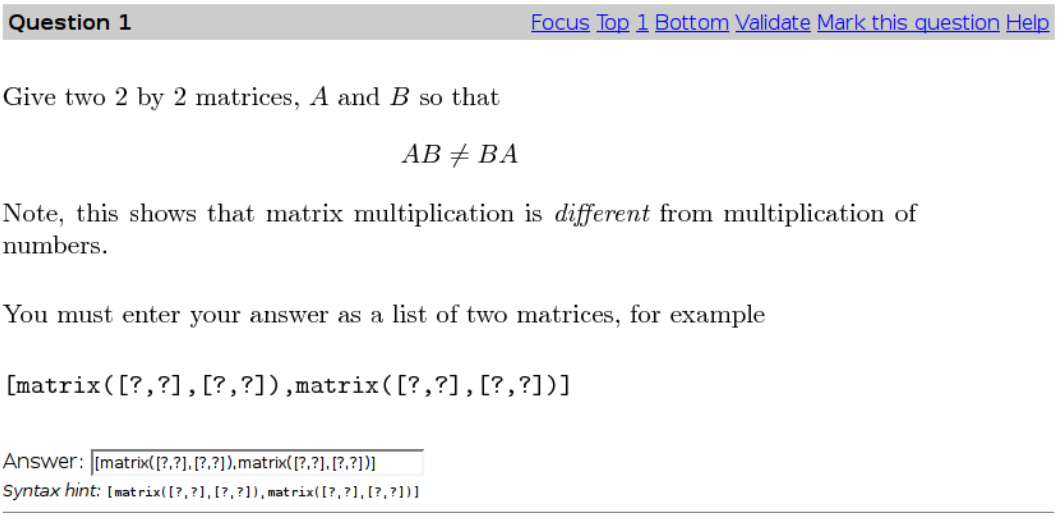

Figure 7.1: Input of multiple matrices to a single text field may require some guidance. This case could benefit from more intuitive user-interface (Fig. 7.2).

commutative ($\mathbf{AB} \neq \mathbf{BA}$), (Fig. 7.1). In STACK version 1.1 this exercise requires the student to give a list of two matrices on a single line. The student is instructed with the following syntax hint:

```
[matrix([?,?],[?,?]),matrix([?,?],[?,?])]
```

From the student's point of view this form of input is not very intuitive. While most people are able to correctly input matrices into a single text-field, inputting multiple matrices or any other type of values is not so simple.

In this example the two matrices are given as a list and decoded as two separate matrices at the feedback variable definition phase. Basically the only phase where the two matrices are required to be joined together as a list is when STACK reads the input and sends it for validation and marking. Basically, we can map two separate inputs to a single vector before we give it to STACK, and then extract them from the vector before we show them to the user. It is more difficult to direct feedback to correct input field and to render the validation messages correctly. For these tasks we need some hooks in the feedback and validation code.

The most important modifications required for multiple inputs are not the additional input fields but the fact that in the case of multiple fields we need to label the fields. With these labels we may give silent syntax hints. Consider the question where one is required to give a function $f(x)$ that has certain properties or matches

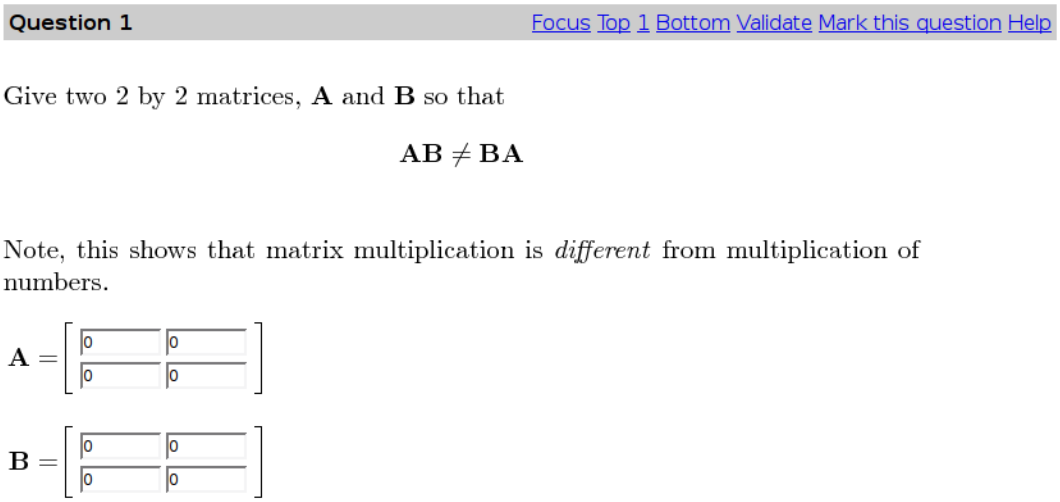

Figure 7.2: Giving matrices and especially multiple matrices is probably simpler in these input fields than to a single text field.

to a given picture. If this exercise has an input-field labelled "*Answer:*" one can expect that most answers are either of the form `x^2` or `f(x)=x^2`. While in this case one can test for both of them, it is still a problem for the student to decide what should be the input. On the other hand, if the input field is labelled "$f(x) =$" there is no ambiguity about the form of the answer, the student just needs to fill in the blank.

The implementation of multiple inputs changes the storage type of the variable `studentAnsKey`[1] of the question prototype to a comma separated list and stores the teacher answers as a list. For labels a new variable is needed in the question prototype. The variable stores a list of labels as a `castext`[2] type variable that gets a special treatment from STACK. When the question is instantiated, the comma separated list gets transformed as markers for rendering hooks and the questions feedback variables get mapped to input fields. Basically, we just replace the instantiation code with a version that identifies the modified storage type and translates it to markers that STACK does not notice and to parameters that STACK does not have problems with.

[1] The variable storing the name of the variable to which the input is stored.

[2] A data type defined in STACK code, consisting of LaTeX-code that may contain variables from the CAS.

7.1.3 2D input

While the example in the previous section is quite acceptable with two separate input fields, it would probably work even better if the matrices were given into a matrix of text-fields. Matrices, vectors and lists are not the only answer types that could benefit from specialised input controls:

- Boolean values are obviously best handled with check-boxes. One could always write "1" or "0" or "true" or "false" to a text-field, but check-boxes are simpler. Unfortunately, check-boxes are problematic from STACK's point of view because it cannot tell whether the question has been answered or not with standard check-boxes. Is an unchecked check-box is really an answer signalling "false" or just an unanswered question? For this problem the obvious solution is a three state check-box that also has the value "undefined" available and which is by default active. Unfortunately, the three state check-box is not available as a standard HTML form element and therefore cannot be used.

  Due to the problems related to identifying unanswered questions one needs to write special grading algorithms when using check-boxes, as one cannot be sure that the student actually answered to the question.

- Selecting one from many is naturally suitable for radio buttons but of course we may use drop-down menus. Naturally, this could be handled with a text-field, but again writing the selection to a text-field probably leads to errors.

  Drop-down lists and radio buttons may also be used to emulate three state check-boxes, at the expense of screen space.

- Selecting many of many consists of many boolean choices. Therefore it suits perfectly for check-boxes, because writing a list of selections to a text-field would again be a source of input errors. The problem with check-boxes is a problem also in this case.

Basically 2D input just replaces the code that is used to render the text-field with a code to render other components. It also replaces the code that reads the submitted values from POST with a code that reads the values from multiple fields and translates them to a form that STACK understands. With selection type input controls it can do some additional randomising.

Following types of input controls have been provided:

Raw Used to input anything into a single text-field.

Matrix/Vector Grid of text-fields, the size and dimension of the grid is automatically matched to the size and dimension of the teachers answer.

Selection 1 of N Selection from value-label pairs with either radio-buttons or drop-down list. Optionally provides "other" field for freeform input, randomises the ordering or selects a random subset from the options for display. When selecting a random subset from the options, the teacher's answer can be forced to be a part of the subset. If not so, the "other" field will be provided.

Check-box Single check-box that returns `true` when checked and `false` otherwise. There exists also a binary check-box that returns 1 or 0 and makes it then easier to do calculations.

Custom layout Allows one to position the input controls freely and write instructions among them. Custom layout has its own syntax for placement of controls. The layout is defined by placing markers to the places where one wants to have the controls. A marker is of a form:

```
--|VARIABLE|TYPE|SETVALUE|--
    --|ans1|Radio|1|--
    --|ans1|Radio|2|--
    --|ans1|Radio|3|--
--|ans2|Matrix|ident(3)|--
    --|ans3|Checkbox|--
    --|ans4|Raw|x^2+2|--
```

The types accepted are:

`Raw` A single text-field, the set value defines the size of the field.

`Matrix` A vector or a matrix, requires a set value to decide the size of the field.

`Checkbox` A single check-box returning `true` or `false`. Check-boxes do not need set values.

`CheckboxBinary` A single check-box returning `1` or `0`.

`Radio` A single radio-button, radio-buttons with the same variable form a radio-button group. The set value of the selected radio-button is the value that gets returned.

7.2 Display

STACK's normal way of displaying equations and other content via TtH is not very good for our target browsers, so we chose to replace it with a more compatible one. Our solution renders images of all the material to be displayed.

The main problem with the target audience browsers is that the TtH renderings of mathematical formulae required special fonts for them to be correctly displayed and as the users did not always have those fonts available, the results (Fig. 3.1) tended to be unacceptable. For this reason we chose to use an approach that does not require any special fonts or features from the target system.

Because the source material is basically just LaTeX-code, normal LaTeX-interpreter was used to turn it into a normal document. That document is then converted to a high resolution bit-map image and trimmed so that the margins get removed. The resulting image is then displayed to the user and as there should not be any problems in displaying bit-map images all browsers should be able to display the exactly same output.

For the sake of efficiency we store all generated images for reuse so that we can avoid the expensive generation process if the same source code needs to be rendered again. For this reason each generated image is stored with the hash of the source code. Before creation of an image we first check if there already exist an image with the same hash. Actually, the images are tagged with two different hashes of the source code to lessen the likelihood of hash collision.

7.2.1 Form element injection to images

The custom layout option that allows placement of input fields among normal text and/or mathematical layout means that the image that displays the whole layout must also display the form elements at the places defined by the layout. As images of form elements are not actually interactive form elements we need to place the actual form elements on top of those images. To do this we need to do some extra processing:

1. When the document is rendered the LaTeX-code generates colour coded rectangles to mark the spots where the form elements should be.

2. The rectangles are then identified from the generated bit-map by a custom built C++ program that will report the exact positions and sizes of the rectangles for use in the generation of the HTML-code for the form elements.

3. Based on the position and size data a suitable HTML-code will be written for each form element, the positioning is done with CSS-layout.

Chapter 8

Test applications

8.1 Writing exercises

While STACK can do many things it still has all the limitations of a bounded input CAA system, i.e. one can only ask for specific values and only those very specific values [8]. But this testing can be done in many ways and with STACK's grading logic it is based on the grading tree. There is basically no other limit than the author's ability to handle a large decision tree.

In the following sections we examine some possible ways of posing questions in the cases of two slightly different courses.

8.1.1 Basic exercise guidelines

Keep it symbolic

STACK is a system that evaluates symbolic answers [7], and although it is possible to use it for multiple-choice questions and questions involving float values, it would waste the system's capabilities.

Reverse approach

Building parametric exercises tends to be simpler, if the exercise can be reversed i.e. if you can calculate the initial values from the result. With STACK you have a CAS in your use and it can easily evaluate most inverse operations. One should always try to build the exercises with the reverse approach as it:

- ensures answers that have a specific form and that can be presented easily. Simple initial values can lead to complex results but with simple result the initial values are generally reasonably simple,

- typically generates similar exercises provided that the randomised results do not vary too much. There might still exist some problems in handling numbers or terms in some cases,
- generates reasonably simple coefficients for the initial values if the result coefficients are simple i.e. most (if not all relevant) operations resulting in integer coefficients start with rational coefficients or fractional exponents.

Give an example of something

With a CAS based CAA system like STACK it is easy to check the properties of the answer and this possibility should be used as much as possible. Instead of just posing questions that require a specific unique answer one should also build exercises that check the answer for specific properties. In other words one can now ask questions of the form "give an example of" and, as the CAS evaluates the submitted answers, one does not need to limit these exercises to simple cases that are easy to check manually.

The fact that one can check the properties of the answer can also be used for grading. Consider, for example, the following exercise:

- The student is given a set of N points and then asked to fit a polynomial of any degree to the points so that it describes the data.
- The grading algorithm evaluates the RMS for the polynomial and checks its degree. It may also check the RMS for a hidden validation set of data-points.
- If the polynomial is too high of a degree or too low, then the grade is lowered and the student gets feedback about over-fitting or not representing all features.
- The RMS affects the grade directly; when it rises over some threshold values the grade drops.

Naturally, the data sets may be generated from a random function. The student could also be given more fitting options than just polynomials, i.e. the student could construct whatever function he or she wishes as long as the underlying CAS can evaluate it.

Forbidden words

STACK evaluates the submitted answers as CAS expressions. This gives the student the possibility to make the CAS to do the exercise, i.e. instead of inverting some

matrix or integrating something the student can return a CAS expression that does the job. For this reason one should make sure that these kinds of CAS commands are listed as forbidden commands.

While the "black-list" approach of STACK to this problem might seem odd, as it would probably be simpler to build a "white-list" of allowed commands, there is a perfectly valid reason for it. Consider a situation where one uses trigonometric functions in the answer. What if STACK only accepts sin, cos and tan, as these are the functions the student would need? Now what happens if a student decides to use cotangent instead of $\frac{1}{\tan \alpha}$? Would you want to instruct the students not to use that function, and how could you predict all the possible ways to input some answer?

8.2 Control engineering exercises

The Basic Mathematics for Control Theory course trains students in the use of certain basic mathematical tools needed in basic control engineering. While the course does explain part of the theory behind the methods the focus is in training the methods and actually applying them. From this point of view the exercises for this course should give the students the ability to test their skills repeatedly against similar exercises. Therefore this is the perfect place for parametrised exercises. Additionally, the exercises should focus on specific methods and make sure that the student knows which methods.

8.2.1 Example 1. Laplace transform tables, repetition

The basic problems involving Laplace transforms that the students should get used to, and training with them is a very basic type of exercise. With STACK one would build a exercise like this:

1. First one decides whether the inverse transform is going to be needed or not. This can be done by a random variable, but generally it is a static decision.

2. Then you pick some of the currently trained transforms from a table at random. (Fig. 8.1)

    - One can pick multiple ones and sum them together, if one wants to make a bit more complex exercise.

3. Then one randomises some (integer or symbol like π) values for the result of

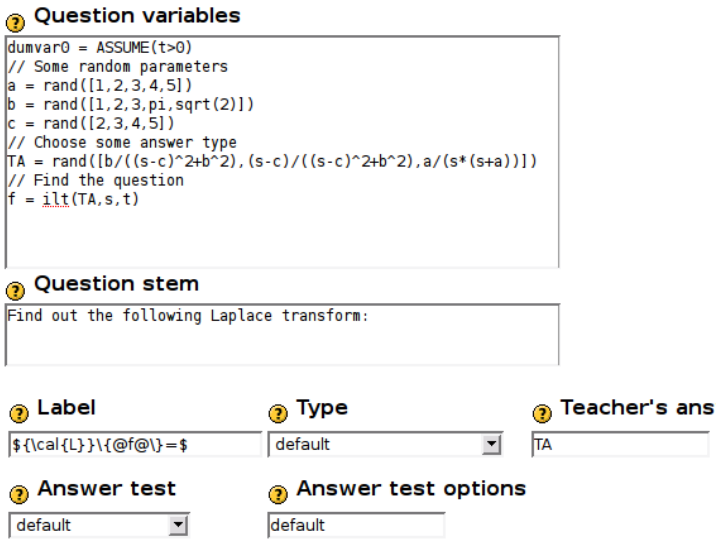

Figure 8.1: An example of exercise initialisation code that picks a random answer type from a list and then uses an inverse operation to find out the value(s) to give to the student so that the result is of the chosen type. The function `ilt` is naturally the inverse Laplace transform.

the transformation, i.e. if we train the basic transformation we want the result to look nice in the frequency space or vice versa.

4. One then gets the software to calculate the opposite transformation, to generate the function for the student to transform.

   - Of course, when we have the tables we could use them to generate it.
   - One can then make things more interesting by simplifying the function, that way the student needs to for example do partial fractions.

5. The grading algorithm just compares the result with the correct one. But if one wishes to check some special cases, that is always possible.

In principle one can build an exercise template to be reused multiple times to train some very specific transform with slightly different parameters, and maybe mixing some other component to the function for distraction. And the student can try doing the exercise as many times she or he wishes.

Traditionally this kind of exercises have been given as a bulk set in the form "transform functions a–f". With STACK this kind of presentation is not the best way, because showing so much information on single screen with input fields is not

a good idea. Additionally, handling the grading of multiple "optional" inputs is complex, as you need to consider cases where the student only answers to some parts of the question now, and should not be penalised for the empty parts before he or she submits values for them.

8.2.2 Example 2. Calculating matrix exponential, with intermediate steps

Calculating the matrix exponential in symbolic form is a common and complex task, mainly encountered when discretising state space models. The problem is that in exercises requiring the calculation of the exponential, the crux of the problem is in the way it is solved, i.e. we need to know that both the solution and the intermediate steps are correct. The student is then required to give matrix values, and we should have as few of them as possible to avoid too complex forms for submitting the solution. And as a minor complicating factor there are many ways of calculating the exponential [23].

A way this question could be built is as follows:

1. Generate a matrix to be operated. Randomising a reasonable result and then solving the start point might not be enough here and some fine tuning and heuristics might be required to build a suitable answer matrix or initial values. In this case it might be a good idea to ensure that some elements are zeros to make the exercise technically easier to calculate. Checking whether the matrix is nilpotent could also be a good idea.

2. Ask the student to calculate the exponential and to choose the method he or she uses from a short list. Ask for some method specific intermediate values like eigenvectors, some part of a decomposition or, for example, the inverted matrix (in the inverse Laplace transform method). (Fig. 8.2)

    - Students could be encouraged to test some methods by giving more points for them.
    - The student would not get any points if the intermediate step was wrong or missing, When an intermediate step is correct but the result is not it is probably a good idea to give some points.

3. The grading algorithm just compares the intermediate and final answer to the correct correct ones and gives the grade.

Question 1 Focus Top 1 Bottom Validate Mark this question Help

Give the matrix exponential of the following matrix:

$$\mathbf{A} = \begin{pmatrix} \pi & 5 \\ 0 & 2 \end{pmatrix}$$

Use one of the listed methods to calculate it, tell us which method you used, and give the corresponding intermediate value. Note that some methods and intermediate values are harder than others. Note also that if the intermediate value is wrong you will not get any points.

$e^{\mathbf{A}} =$ [0 | 0 ; 0 | 0]

Method:
- Inverse Laplace ($\mathcal{L}(e^{t\mathbf{A}}) = (s\mathbf{I} - \mathbf{A})^{-1}$), give $(s\mathbf{I} - \mathbf{A})^{-1}$ as the intermediate value.
- Diagonalisation, if $\mathbf{A} = \mathbf{V}\mathbf{D}\mathbf{V}^{-1}$ then $e^{\mathbf{A}} = \mathbf{V}e^{\mathbf{D}}\mathbf{V}^{-1}$, give $\mathbf{V}$ so that $\mathbf{D}$ is diagonal.

The intermediate value: [0 | 0 ; 0 | 0]

Figure 8.2: A way for checking that the solution has been reached using some specific method. One can have as may methods as desires as long as these methods have a testable or distinctive intermediate value. Here the first option has just one correct intermediate value while the second one accepts all matrices that fulfil the requirements.

The problem is that this procedure only works with exercises that do not require anything more than matrix exponential as mixing other actions could make evaluating of the result complicated as the number of input fields increases.

8.2.3 Example 3. Controller design, interactivity by changing the grading

This example is more of a technology presentation than a real exercise for this course. The exercise could be given as follows. The student is shown a impulse or step response of a simple second order system. He/she is then asked to give a transfer function matching it and a controller for it, so that some design parameters are met. How this could be done is:

1. Generate the system by picking random zeros from a suitable place. To make perfect answers/guesses improbable and the exercise more "realistic" one should avoid integer values.

2. Plot the system for the student with the input fields for both the system transfer function and the controller.

3. When a student gives his/her answer we would plot a new response displaying the real response next to the students system, and maybe another plot with different input. And if the controller was given we would also plot the response for both the real system and the students model using that controller.

4. In this case the grading would probably be lenient as the design parameters would probably require couple of tries before a good enough controller would be found. The penalty for failed attempts could even be completely removed or at least be very small.

    - If so desired we could grade the model based on the distance of its zeros from the real zeros. But the grading of the model would not be very interesting in this case anyway.
    - Each design parameter could have multiple different levels, i.e. if you cannot get your systems settling time low enough, but the rise time is much better than required you could get more points from it to compensate the lost points.

This case could be turned to a graded exercise with penalties that would limit the number of tries, i.e. every test run after the first costs 10% of the maximum points.

8.3 Basic mathematics exercises

On Basic Course in Mathematics KP3-I students are introduced to the basics of complex analysis among other things. The course has a more theoretical goal of course but it still has plenty of simpler operations to be trained.

8.3.1 Example 4. Need for multiple input fields

As a simple exercise the student is required to split a complex valued function to real and complex parts:

$$f(z) = u(x, y) + \mathrm{i}v(x, y), \qquad z = x + \mathrm{i}y.$$

CAS can easily solve this for various random functions $f(z)$ and compare the results to those given by the student. But what STACK cannot easily do, is to compare single line result of the form $u(x, y) + \mathrm{i}v(x, y)$ to the answer, because, while it can extract the real and imaginary part of any expression, it cannot easily check if they have already been separated. For this reason this exercise requires two input fields that take the functions separately. One should always remember that while STACK does have some functionality for checking the way something is expressed, there are still many different ways for expressing the same thing. Testing them all may prove to be more complex than just to give a couple of blank fields for the student to fill in the missing parts.

8.3.2 Example 5. Simplest ways are just as good as more complicated ones

Parametrisation and using the CAS to generate and solve the exercises may be fast and seem like the next big thing, but they are not required for creation of exercises. One may use STACK without writing a single line of exercise initialisation code by just giving the answer to compare with the students answers (Fig. 8.3). There are some obvious benefits with this approach:

- It is the simplest way to start using STACK.
- It is fast to do.
- There is very little that could go wrong.
- It is so easy to do that one can hire a student to do only the hard part i.e. transferring the questions text to the system and then just going trough the

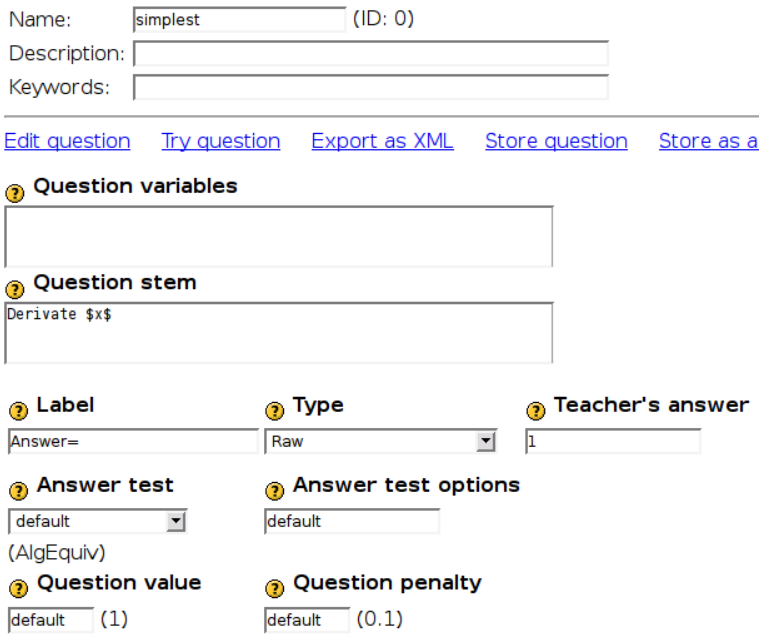

Figure 8.3: Minimum amount of coding is required if one does not need special checks for grading.

exercises and add the answers.

Naturally there are additional things that are not that easy to do. For example, this approach only grades exercises in a binary way, without additional grading logic and it can neither spot partially correct answers nor can it give the user any feedback.

This approach suits also perfectly for multiple choice exercises. Other typical exercise types are geometric exercises where the student for example has to give the volume of some object described in the text or as an image, or exercises asking for some other values based on properties of the described situation/object.

Chapter 9

Conclusions

9.1 Technology

We have seen that, in the technical sense, CAA is mature enough for serious use. And once certain presentation problems are solved there should not be any problems with its use. While there are problems with presentation of complex mathematical formulae, this should not discourage anyone from starting to use CAA software. At first one should work with simpler concepts, and then as presentation technologies like MathML develop, expand to more complex fields.

Those worried to start using CAA software now due to fears of losing the work done when transferring to next versions it should be noted that the real work done with CAA goes to the grading algorithms, and they tend to be universal in nature. That is, it is unlikely that any future CAA software would be unable to handle for example decision tree based grading algorithms.

9.2 Future expectations

It is highly likely that CAA will gain serious foothold in basic courses and as a recap material for basic concepts on more complex courses. And as the teachers gain experience with the technology they will start to experiment with more complex exercises.

In any case it is unlikely that CAA would become a tool for small specialised courses before a common CAA software gains large enough user-base and sharing of exercises between teachers begins. But once there are enough teachers using CAA, the likelihood of finding enough ready made exercises for use in a specialised course increases and should eventually reach the point where even these courses will have

enough ready made exercises to start using CAA. As new application fields start using CAA the systems themselves will also be developed to better support these new fields. With open source CAA systems the development can happen rapidly and reach some really exotic fields.

9.2.1 High quality evolving exercises

Once the basic exercise types have been created in parametrised form, the evolution of the grading algorithms begins as teachers start to add checks and feedback for common errors to them. In few years an exercise might gain dozen or so new special case checks and more and more students get feedback on what went wrong.

The only problem is that once these exercises start to circulate it is nearly impossible to collect new modifications back to the source version. Eventually the exercises will evolve to such level that people will want to build exercise banks (or more likely grading algorithm banks) to store them.

9.2.2 Final words

CAA is here to stay and although it will still take years for superior quality exercises that have feedback for every imaginable error to emerge, they are still coming. It is unlikely that there will be exercises with grading algorithms that have more than fifty checks in a couple of years. But as the more complex check structures will migrate in to the system to be base level system provided checks, thus the algorithms will grow by simply abstracting away the underlying checks.

What now takes dozens of tests tied together will no doubt soon be available as simple prepackaged components e.g. a component to check if the answer is a general solution for a differential equation. And eventually one will have stacked together dozen or so components and notices that those components together check for nearly fifty different cases.

I for one welcome our conscious grading algorithm overlords/.

Bibliography

[1] Kauko Hämäläinen and Ritva Jakku-Sihvonen. More quality to the quality policy of education, August 31, 1999. ISBN 9521308893. Background paper for the Meeting of the Ministers of Education.

[2] Christopher J. Sangwing. New opportunities for encouraging higher level mathematical learning by creative use of emerging computer aided assessment. *International Journal for Mathematical Education in Science and Technology*, 34 (6):813–829, 2003.

[3] Gustav W. Delius. Conservative approach to computerized marking of mathematical assignments. *MSOR Connections*, 4(3):42–47, August 2004. ISSN 14734869.

[4] Terry Anderson and Fathi Elloumi, editors. *Theory and Practice of Online Learning*. Athabasca University, 2004. http://cde.athabascau.ca/online_book/ (referenced 5 March 2008).

[5] Grant Keady, Gary Fitzgerald, Greg Gamble, and Christopher Sangwin. Computer-aided assessment in mathematical sciences. In *Proceedings of the UniServe Science Conference, 'Assessment in Science Teaching and Learning'*, pages 69–73. UniServe Science Uni of Sydney, September 28, 2006. ISBN 1864878657.

[6] Christopher J. Sangwing. Assessing mathematics automatically using computer algebra and the Internet. *Teaching Mathematics and its Applications: An International Journal of the IMA*, 23(1):1–14, March 2004.

[7] Christopher J. Sangwing. Assessing elementary algebra with STACK. *International Journal for Mathematical Education in Science and Technology*, 38(8): 987–1002, August 27, 2007.

[8] Christopher J. Sangwing. STACK: making many fine judgements rapidly. In *CAME 2007 - The Fifth CAME Symposium*, 2007.

[9] Antti Rasila, Matti Harjula, and Kai Zenger. Automatic assessment of mathematics exercises: Experiences and future prospects. In *ReflekTori 2007 - Symposium of Engineerin Education*, pages 70–80. TKK: Teaching and Learning Development Unit, 2007.

[10] Aki Hiisilä. Course management system for basic courses in programming (in Finnish). Master's thesis, Helsinki University of Technology, May 2005.

[11] Ari Korhonen, Lauri Malmi, and Panu Silvasti. TRAKLA2: a framework for automatically assessed visual algorithm simulation exercises. In *Proceedings of Kolin Kolistelut / Koli Calling – Third Annual Baltic Conference on Computer Science Education*, pages 48–56. Helsinki University Printing House, 2003.

[12] Seppo Pohjolainen, Martti Ala-Rantala, Ossi Nykänen, and Heli Ruokamo. On the design and evaluation of an open learning environment. *International Journal of Continuing Engineering Education and Life-Long Learning (IJCEELL)*, 9(2):249–61, 1999. ISSN 1560-4624.

[13] Lee Harvey and Peter T. Knight. *Transforming Higher Education*. Open University Press, 1996. ISBN 033519589X.

[14] Trygve Reenskaug. Models-Views-Controllers. December 1979. Xerox PARC, Technical report.

[15] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Professional, November 10, 1994. ISBN 0201633612.

[16] Pavi Sandhu. *The MathML Handbook*. Charles River Media, 2003. ISBN 1584502495.

[17] Titus Winters and Tom Payne. Computer aided assessment with human oversight. In *ITICSE '06: Proceedings of the 11th annual SIGCSE conference on Innovation and technology in computer science education*, page 320. ACM, New York, NY, USA, 2006. ISBN 1595930558.

[18] André Heck. Assessment with Maple T.A.: creation of test items. http://www.adeptscience.co.uk/products/mathsim/mapleta/MapleTA_whitepaper.pdf (referenced 5 March 2008), November 29, 2004.

[19] Michael Gage, Arnold Pizer, and Vicki Roth. WEBWORK Generating, Delivering, and Checking Math Homework via the Internet. In *ICTM2 International Congress for Teaching of Mathematics at the Undergraduate Level, Hersonissos, Crete Greece*, 2002.

[20] Barry Simon. First looks: Animate your equations with Macsyma. *PC Magazine*, 14(2):56. ISSN 08888507.

[21] Simon Josefsson. RFC 4648: The Base16, Base32, and Base64 Data Encodings, October 2006.

[22] Tim Berners-Lee, Roy Fielding, and Henrik Frystyk. RFC 1945: Hypertext Transfer Protocol — HTTP/1.0, May 1996.

[23] Cleve Moler and Charles Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.

Appendix A

Grading algorithm examples

Building grading algorithms with the decision tree structure of STACK may require that one uses many test more than when using a specialised algorithm structures for special tasks, but the tree structure does have its own benefits. Mainly, the practically infinite number of tests that can be queued in the tree and the simplicity of just dividing the answer in two distinct sets at each level of the tree. One important aspect of the tree structure is that one may easily add new tests to it and thus just refine the previous classifications to new subclasses of answers.

In this chapter we will give examples of the basic tree structures for specific test types. For graphical presentation of the tree, figures like A.1 are used. In these figures certain terms and concepts are presented in condensed form. At the top of the figure the question will be displayed in its own box as well as all the inputs and possible parameters, after them possible preprocessing operations will be listed before the tree. The tree will be shown with the root-node first, each node will contain a test to be evaluated. Arrows leaving from the left side of a node correspond with failed tests and the those leaving from the right side with passed tests.

A.1 Exercise 1: the simplest tree

The simplest exercises (Fig. A.1) can just compare a static value with a value given by the student. In these cases the grading algorithm is typically just one node testing if the values are equivalent.

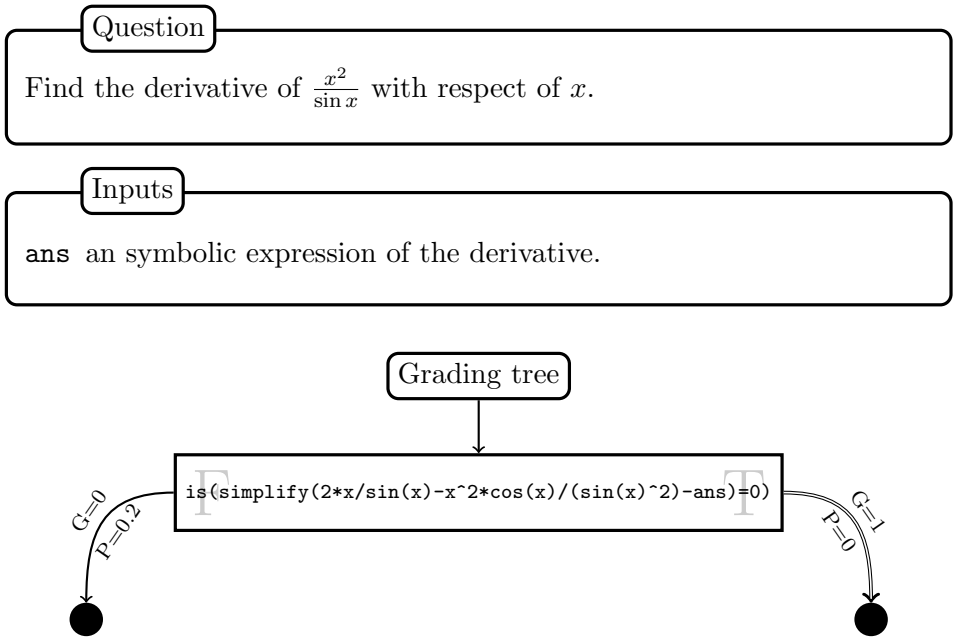

Figure A.1: Simple exercises do not require complicated trees, here we only have one node in the tree. Note that in this case storing the teacher's answer to some variable is recommended, as writing it into tree node does bloat the three with unnecessary amount of data. Typically the teacher's answer is stored to a variable named `TA` and the student's answer to `SA`.

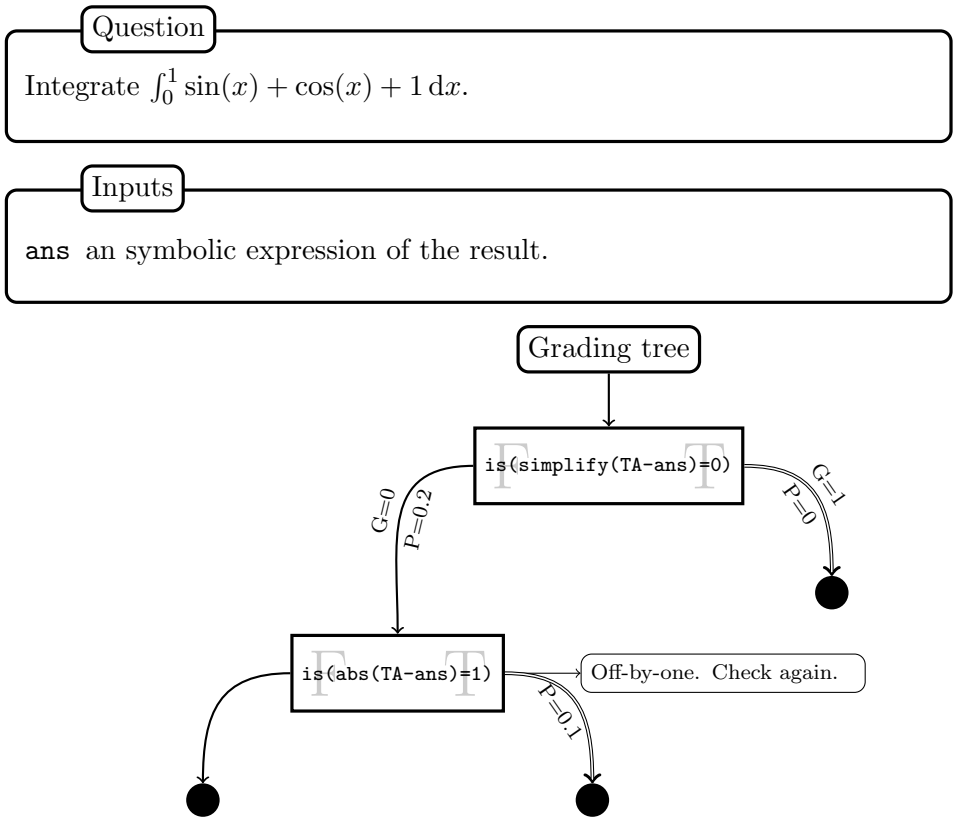

Figure A.2: Error identification is just simply checking for other possible answers, and adjusting the grade based on them. Here `TA` has been omitted to save space.

A.2 Exercise 2: error identification

When we know that there is a common error, that can be made and that leads to a specific incorrect answer, we can add a test for that answer. That test will be placed in the branch of the tree handling the error case (Fig. A.2) or the unknown error if we already identify other errors.

A.3 Exercise 3: answer preprocessing

The tree may not be the most practical place to handle all tests, especially if those tests require comparison of multiple different values. While all test can be written open so that they can be placed directly to the nodes of the tree not all of them should be, as the legibility of the algorithm will most likely suffer. In most cases one should evaluate all but the simplest tests in the preprocesing phase and store

the results as numeric or boolean values that can be easily used to route (Fig. A.3) the algorithms process through the tree.

A.4 Exercise 4: feedback

The nodes of the tree can give feedback based on the test results, the only limiting factor is that the feedback will be rendered in the order the tree is walked trough thus one may need to think where one places the feedback. In most cases the feedback should be kept as atomic as possible so that feedback coming from some test on the route does not seem odd when later test append to it. If one has built a true tree (a tree where each node can only be entered from single node) one may write the whole feedback to the leafs (as there is only one way to reach those nodes) of the tree, but in most cases some of the nodes may connect to both of their parents outputs (Fig. A.4).

A.5 Handling points and penalties

Each node of the tree may adjust the points and penalties given for the answer as they wish based on the result of the test. Typically, one increases the points when tests signal a correct answer or reduces the points if there is something wrong in the answer. The penalty is typically increased only when one spots serious errors (or attempts to cheat) one can also decrease or completely remove the penalty in cases of less serious errors or when the error is not the students error e.g. if the instructions were not exact enough (by purpose or not) one may give more instructions as feedback and drop the penalty at the same time.

Points define the number of points one gains if this was the first attempt to answer to this exercise and penalties define the amount of points subtracted from the maximum points of the following attempts. The exact effect of the penalties depends on the grading system used, in some cases there are no penalties and in some cases the student may only try once.

In the first example (Fig. A.1) the points and penalties are just defined based on the result of the only test as there are no other tests to change the values, but in the second example (Fig. A.2) the second test sets the penalty to a different value. This method of setting the values explicitly always when one wishes to change can be useful when dealing with extreme cases but when one has many factors affecting the values like in example (Fig. A.4) one may wish to use incremental ways of modifying the grade. With incremental modification handling complex structures

Question

Find the roots of `@p@`= 0, give your answer as exact values in the form $a + b$i.

Variables

```
a=rand([-3,-2,-1,1,2,3])+i*rand([-3,-1,2,4])
b=rand([-3,-2,-1,1,2,3])+i*rand([-4,-2,1,3])
p=expand((z-a)*(z-b))
```

Inputs

`z1, z2` symbolic expressions of the roots.

Preprocessing

```
z1ok=if(ev(p,z=z1,fullratsimp) = 0) then 1 else 0
z2ok=if(ev(p,z=z2,fullratsimp) = 0) then 1 else 0
diffok=if(z1=z2) then 0 else 1
allok=z1ok*z2ok*diffok
```

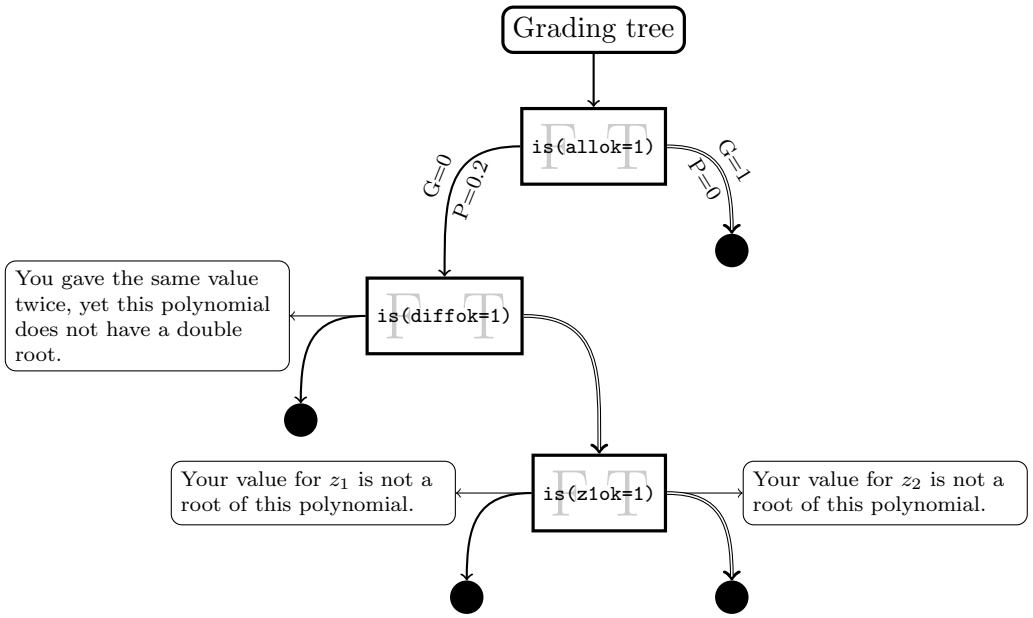

Figure A.3: Preprocessing will help when the tests used in the nodes get complicated. In this case we have the problem of handling the order of the answers, as one may give the roots in two different orders.

Question

Find the trace, rank, and determinant for the following matrix: @m@

Variables

```
n=rand([3,4,5])
m=rand(7+zeromatrix(n,n)) // elements between 0 and 7
```

Inputs

`t,r,d` the trace, rank, and determinant of the matrix.

Preprocessing

```
tok=if(trace(m)=t) then 1 else 0
rok=if(rank(m)=r) then 1 else 0
dok=if(determinat(m)=d) then 1 else 0
allok=tok*rok*dok
allbad=(1-tok)*(1-rok)*(1-dok)
```

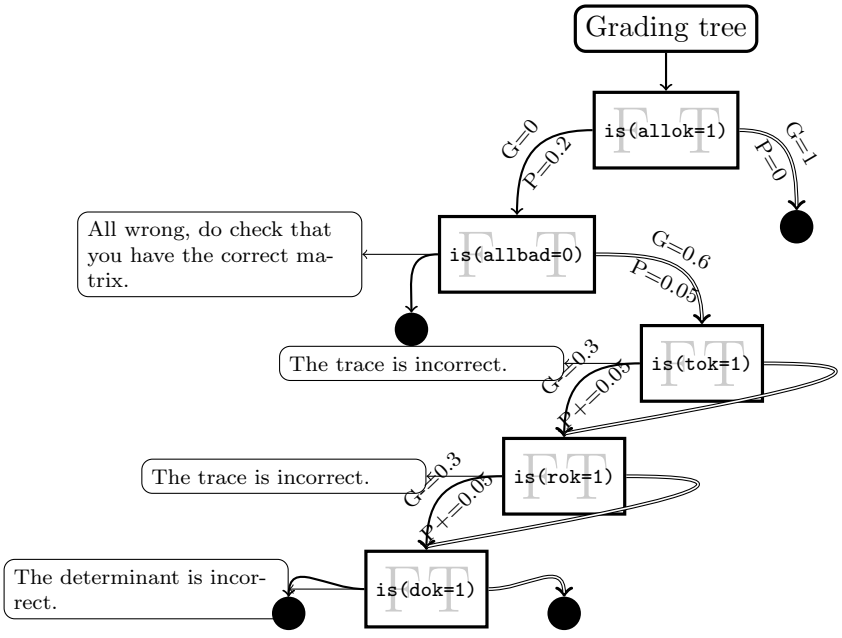

Figure A.4: Giving feedback may require some additional ordering of the tests as the feedback is outputted to the same output stream. Here the two first nodes are just for specific feedback.

may be simpler as long as the values do not start to sum up to much, although no one has said that they may not rise over 1. Though a penalty of over 1 i.e. over 100% may be a bad thing.

Appendix B

Source code for the user interface modifications

B.1 Structure

The modifications made to the system are basically transformations from the systems own representation format to slightly more verbose format and back, basically. These transformations always happen at the rendering phase, in other parts of the system all additional data required for the modifications travels tied to the normal question related data but staying inert.

The new functions and data are tied to the old data representation by building a class that encompasses all the necessary parts (listing B.1). Calls to relevant functions in STACK's code are then changed to calls to this classes implementations of those functions and few hooks are added to gain access to Maxima for rendering and randomisation.

The input fields are also represented as an object (listing B.2) joining the data and functionality.

```
1 class QuestionInstance {
2   // The input fields
3   var $inputs = false;
4
5   // The labels for fields are rendered and cached
6   var $labels = false;
7
8   // The display type is stored as it can not be accessed otherwise
9   var $disp = '';
10
11  // The constructor parses the fields and stores them
```

```
12  function QuestionInstance(& $questionInst, $disp = NULL) {
13    if ($disp != NULL)
14      $this->disp = $disp;
15    if (array_key_exists('QI', $questionInst)) {
16      // No need to redo things
17    } else {
18      $this->inputs = modinput_parser($questionInst);
19      $questionInst['QI'] = $this;
20    }
21  }
22
23  // Based on the $disp variable we render fields to HTML
24  function fieldtoHTMLFormFrag($field) {...}
25
26  // Sets the value(s) from maxima formed input, this function the
27  // multiple inputs back from the internal representation format
28  function setRawAns($ra) {...}
29
30  // This one is used to read back the values requested from maxima
31  // during intialisation. Including labels and random parameters.
32  function readSpecialLocals($specials, $options) {...}
33
34  // Reads the raw answer from a source (basically from the input
        fields)
35  // and "encodes" them to a single vector for use inside STACK
36  function getRawAns($from = 'POST') {...}
37
38  // The following functions (stack_question_* => question_*) are
        mostly
39  // just  wrappers for the original functions. Basically, they are
        like
40  // this one but may include some additional data transformations.
41  function question_inst_mark(& $questionInst, $options, $RawAns, &
        $errors) {
42    return stack_question_inst_mark($questionInst, $options, $RawAns,
          $errors);
43  }
44
45  // Most of the output functions need some tuning and this one is
46  // a complete rewrite
47  function question_inst_feedback(& $this_attempt, $mark, $options,
        $errors) {...}
48
49  function question_inst_displayq($qInst) {...}
50
```

```
51    // This one builds the frame for the fragments coming from
52    // fieldtoHTMLFormFrag
53    function question_inst_try_formfrag($qInst, $lastans, $ffn,
          $further_attempts) {...}
54
55    // Slightly modified layout
56    function question_inst_try_form($questionInst, $lastans, $PostTo = ''
          ) {...}
57
58    // Slightly modified layout
59    function question_inst_show_sol($questionInst, $errors) {...}
60
61    function question_inst_show_attempts($questionInst, $errors) {...}
62
63    // Slightly modified layout
64    function question_inst_try_test($questionInst, $this_attempt,
          $options, $errors) {...}
65 }
```

Listing B.1: The question instance object that is used to combine STACK's normal question instances to the new input fields that are extracted from the normal fields (line 18). Note that the object is actually stored inside the normal versions associative array (line 19).

```
1  class InputField {
2    // Id of label next to the field, used to pull rendered version from
3    // cache
4    var $label = NULL;
5    // Type of field, for validation purposes (NOT IN USE):
6    // 1 = free        2 = scalar
7    // 3 = vector      4 = matrix
8    // 5 = list
9    var $fieldType = 1;
10   // Form types, for rendering:
11   // 1 = textfield          2 = textarea
12   // 3 = textfield grid     4 = checkbox
13   // 5 = radio button       6 = radio button list
14   // 7 = select
15   var $formType = 1;
16   // More abstract parameters (rows,cols,randomize...)
17   var $fieldParams = false;
18   // Target variable
19   var $ansKey = false;
20   // Order num for multiple fields of a single question
21   var $orderNum = -1;
```

```
22    // Current value
23    var $value = '';
24    // Teachers answer with rendering
25    var $ta = array ('display' => '','value' => '');
26    // Initial value (NOT IN USE)
27    var $init = '';
28    // The question for access to data
29    var $questionInst = false;
30    // Generic size parameter
31    var $textFieldSize = 30;
32
33    // Extracts the value of this field from POST (or GET)
34    function parseFrom($array = 'POST') {...}
35
36    // Parses values from a Maxima list checks for balanced
37    // parenthesis stores value and returns the remainder of the
38    // list once suitable comma has been found.
39    function parseFromMaximaList($stream) {...}
40
41    // Uses high level heuristics to scale the input fields to the
42    // length of the teachers answer... strlen(TA)/#fields+constant
43    function fitToTA() {...}
44  }
```

Listing B.2: The input field class consist of the necessary parameters to define a field and is little more than a struct to store those parameters.

B.1.1 Parsing

The modification relies heavily on wrapping multiple values to vectors/lists for transmission to Maxima, and then parsing them back from those structures. The fact that the values may contain deeply nested structures, like lists of matrices or vectors as well as lists of lists, makes parsing elements from those structures slightly more challenging. For this task a tokenizer function was needed (listing B.3).

```
1  function modinput_tokenizer($in) {
2    $braceCount = 0;
3    $parenthesisCount = 0;
4    $bracketCount = 0;
5    $out = array ();
6    $current = '';
7    $unPlaced = 0;
8
9    for ($i = 0; $i < strlen($in); $i++) {
10     $unPlaced++;
```

```
11      $char = $in[$i];
12      switch ($char) {
13      case '{' :
14        $braceCount++;
15        $current .= $char;
16        break;
17      case '}' :
18        $braceCount--;
19        $current .= $char;
20        break;
21      case '(' :
22        $parenthesisCount++;
23        $current .= $char;
24        break;
25      case ')' :
26        $parenthesisCount--;
27        $current .= $char;
28        break;
29      case '[' :
30        $bracketCount++;
31        $current .= $char;
32        break;
33      case ']' :
34        $bracketCount--;
35        $current .= $char;
36        break;
37      case ',' :
38        if ($bracketCount == 0 && $parenthesisCount == 0 && $braceCount
            == 0) {
39    $out[] = $current;
40    $current = '';
41    $unPlaced = 0;
42        } else
43    $current .= $char;
44        break;
45      default;
46      $current .= $char;
47      }
48    }
49
50    if ($unPlaced > 0 && $bracketCount == 0 && $parenthesisCount == 0 &&
        $braceCount == 0)
51      $out[] = $current;
52
53    return $out;
```

```
54 }
```

Listing B.3: This tokenizer parses a comma separated list of objects that may themselves contain comma separated lists and returns the objects as an array.

B.1.2 Mapping from extended language

The mapping of the extended syntax back to the format STACK uses is done in the function of listing B.4. This function parses the extended syntax, transfers it to the new data structures, and then generates code that STACK understands for collecting and injecting required information to and from STACK. The input field specifications are handled in the function of listing B.5.

```
1  function modinput_parser(&$qInst) {
2    $Rs = array ();
3
4    // We add new vars both tot the questionVars and the ansVars, the
5    // questionVars are used to extract information for the rendering
6    // code and the ansVars are used to map the input information to
7    // the defined variables.
8    $questionVars=array();
9    $ansVars=array();
10   $ansVarLoopIter=1;
11
12   for ($i = 0; $i < modinput_extract_field_count($qInst); $i++) {
13     $options = modinput_extract_field_options($qInst, $i);
14
15     $R = new InputField();
16     $R->label = "$i/-1";
17     $R->orderNum = $i;
18     $R->ansKey = $options['AnsKey'];
19     $R->ta = modinput_extract_answer($qInst, $i);
20     $R->questionInst = $qInst;
21     $R->fieldType = 1;
22
23     // We add the teachers answers evaluation to the questionVars
24     // The id used is assumed free.
25     $questionVars[] = array ('key' => 'ta57591C_' . $R->ansKey,
26             'value' => $R->ta);
27
28     switch ($options['ModInputFieldType']) {
29     case 'Raw' :
30       $R->formType = 1;
31       break;
32     case '1D' :
```

```
33        $R->formType = 1;
34        break;
35      case 'Matrix/Vector' :
36        $R->formType = 3;
37        $R->fieldParams['init'] = true;
38        break;
39      case 'Selection' :
40        if ($options['select'])
41  $R->formType = 7;
42        else
43  $R->formType = 6;
44        $R->fieldParams = $options;
45        $R->fieldParams['list'] = '[';
46        $nnn=0;
47        foreach(array_keys($options['selectionOptions']) as $k){
48  if($nnn!=0)
49    $R->fieldParams['list'] .= ',';
50  $R->fieldParams['list'] .=  "[$k,$nnn]";
51  $nnn++;
52        }
53        $R->fieldParams['list'] .= ']';
54
55        // For selections we need to generate the options
56        $questionVars[] = array ('key' => 'options57591C_' . $R->ansKey,
57              'value' => $R->fieldParams['list']);
58
59        // If we need to randomize the order
60        if($R->fieldParams['random']||$R->fieldParams['limit'])
61  $questionVars[] = array ('key' => 'options57591C_' . $R->ansKey,
62          'value' => 'modinput_selection_randomize(options57591C_'.$R->
                ansKey.')');
63        // If we need to ensure the correct answer
64        if($R->fieldParams['ensure']&&$R->fieldParams['limit'])
65  $questionVars[] = array ('key' => 'options57591C_' . $R->ansKey,
66          'value' => 'modinput_selection_ensure(options57591C_' . $R->
                ansKey.',ta57591C_' . $R->ansKey.')');
67        // If we limit the number of options
68        if($R->fieldParams['limit']&&is_numeric($R->fieldParams['limitN'
            ]))
69  $questionVars[] = array ('key' => 'options57591C_' . $R->ansKey,
70          'value' => 'modinput_selection_limit(options57591C_' . $R->
                ansKey.',' . $R->fieldParams['limitN'].')');
71        else if($R->fieldParams['limit'])
72  $questionVars[] = array ('key' => 'options57591C_' . $R->ansKey,
```

```
73          'value' => 'modinput_selection_randomize(options57591C_'.$R->
                ansKey.')');
74        break;
75      case 'Checkbox' :
76        $R->formType = 4;
77        break;
78      case 'CheckboxBinary' :
79        $R->formType = 4;
80        $R->fieldParams['binary'] = true;
81        break;
82      case 'CustomLayout' :
83        $R->formType = 99;
84        $ll = modinput_extract_raw_label($qInst, $i);
85        $R->fieldParams['raw'] = $ll;
86        $R->fieldParams['subfields'] = array();
87        $k = -1;
88
89        while (($k = strpos($ll, '--|', $k +1)) !== FALSE) {
90    $token = substr($ll, $k);
91    $token = substr($token, 0, strpos($token, '|--') + 3);
92    $tmp = explode('|', substr($token, 3, -3));
93    if (!$R->fieldParams['subfields'][$tmp[0]])
94      $R->fieldParams['subfields'][$tmp[0]] = array();
95    $R->fieldParams['subfields'][$tmp[0]]['type'] = $tmp[1];
96    if ($tmp[2])
97      $R->fieldParams['subfields'][$tmp[0]]['ta'] = $tmp[2];
98    if ($R->fieldParams['subfields'][$tmp[0]]['type'] == 'Radio') {
99      if (!$R->fieldParams['subfields'][$tmp[0]]['tokens'])
100       $R->fieldParams['subfields'][$tmp[0]]['tokens'] = array();
101     $v = $tmp[2];
102     if(strrpos($tmp[2],'*')==strlen($tmp[2])-1&&strlen($tmp[2])>1){
103       $v=substr($tmp[2],0,-1);
104       $R->fieldParams['subfields'][$tmp[0]]['ta'] = $v;
105     }
106     $R->fieldParams['subfields'][$tmp[0]]['tokens'][] = array('token'
            => $token,
107                       'value' => $v);
108   } else
109     $R->fieldParams['subfields'][$tmp[0]]['token'] = $token;
110
111   if ($R->fieldParams['subfields'][$tmp[0]]['ta'])
112     $questionVars[] = array('key' => 'ta57591Csubf_' . $R->ansKey . '_
            ' . $tmp[0],
113           'value' => $R->fieldParams['subfields'][$tmp[0]]['ta']);
114       }
```

```
115
116         $ansVarLoopIter2=1;
117         foreach($R->fieldParams['subfields'] as $key => $field){
118     $ansVars[] = array ('key' => $key,
119             'value' => 'ans00[' . $ansVarLoopIter . '][' .$ansVarLoopIter2
                    .']');
120     $ansVarLoopIter2++;
121         }
122
123         break;
124       }
125
126       $ansVars[] = array ('key' => $R->ansKey,
127         'value' => 'ans00[' . $ansVarLoopIter . ']');
128       $ansVarLoopIter++;
129       $Rs[$R->ansKey] = $R;
130     }
131
132     // If we have only one field we unwrap the vector
133     if($ansVarLoopIter==2)
134       foreach($ansVars as $key => $value)
135         $ansVars[$key]['value']=str_replace('ans00[1]','ans00',$value['
              value']);
136
137     // Merge the special new vars to the old ones, note the order of the
138     // merges, always on 'the inside'
139     if (is_array($qInst['questionAnsVars']))
140       $all_locs = array_merge($ansVars, $qInst['questionAnsVars']);
141     else
142       $all_locs = $ansVars;
143     $qInst['questionAnsVars'] = $all_locs;
144
145     if (is_array($qInst['questionVars']))
146       $all_locs = array_merge( $qInst['questionVars'],$questionVars);
147     else
148       $all_locs = $questionVars;
149     $qInst['questionVars'] = $all_locs;
150
151     $qInst['questionAnsKey'] = 'ans00';
152     return $Rs;
153 }
```

Listing B.4: This mapper maps the extended storage format back to the format STACK can handle, hiding the information to its own data structures.

```
1 function modinput_extract_field_options($question, $N) {
```

```
2   global $stackOptions;
3
4   $fieldSpecs = explode(',', $question['questionAnsKey']);
5   if ($question['modInputQuestionAnsKey'] &&
6       $question['modInputQuestionAnsKey'] != $question['questionAnsKey']
            )
7     $fieldSpecs = explode(',', $question['modInputQuestionAnsKey']);
8
9   $fieldSpec = $fieldSpecs[$N];
10  $field = array ();
11  $field['AnsKey'] = 'ans' . ($N +1);
12  $field['ModInputFieldType'] = 'Raw';
13  $field['select'] = false;
14  $field['other'] = false;
15  $field['limit'] = false;
16  $field['random'] = false;
17  $field['limitN'] = 3;
18  $field['ensure'] = true;
19  $field['selectionOptions'] = array ();
20
21  if ($fieldSpec) {
22    //The new version of the fieldSpec uses slashes to separate all
          parameters
23    //And all of the parameters are simple tokens with nothing too
          complex
24    $fieldSpec = explode('\\', $fieldSpec);
25    for ($i = 0; $i < count($fieldSpec); $i++) {
26      if ($i == 0)
27  $field['AnsKey'] = $fieldSpec[$i];
28      else if (array_search($fieldSpec[$i], $stackOptions['
            ModInputFieldType']['values']) !== FALSE)
29  $field['ModInputFieldType'] = $fieldSpec[$i];
30      else if ($fieldSpec[$i] == 'S')
31  $field['select'] = true;
32      else if ($fieldSpec[$i] == 'O')
33  $field['other'] = true;
34      else if ($fieldSpec[$i] == 'D')
35  $field['ensure'] = false;
36      else if ($fieldSpec[$i] == 'R')
37  $field['random'] = true;
38      else if (strpos($fieldSpec[$i], 'L') === 0) {
39  $field['limit'] = true;
40  $field['limitN'] = substr($fieldSpec[$i], 1);
41      }
42    }
```

```
43
44      //The options are stored among the labels, both the values and
            labels...
45      $labelData = explode('-|-|-|-|-', $question['
            questionModInputFieldLabel']);
46      $theLabels = explode('-|-|-|-', $labelData[$N]);
47      if (is_array($theLabels))
48        for ($i = 1; $i < count($theLabels); $i++) {
49    $pair = explode('-|-|-', $theLabels[$i]);
50    $field['selectionOptions'][$pair[0]] = $pair[1];
51        }
52    }
53    $field['AnsKey'] = trim($field['AnsKey']);
54    $field['ModInputFieldType'] = trim($field['ModInputFieldType']);
55
56    return $field;
57  }
```

Listing B.5: This function provides functionality for the mapper.