Technical Report

UCAM-CL-TR-719
ISSN 1476-2986

Number 719

The Intelligent Book: technologies for intelligent and adaptive textbooks, focussing on Discrete Mathematics

William H. Billingsley

June 2008

This technical report is based on a dissertation submitted April 2007 by the author for the degree of Doctor of Philosophy to the University of Cambridge, Wolfson College.

ISSN 1476-2986

The Intelligent Book: technologies for intelligent and adaptive textbooks, focussing on Discrete Mathematics

William Henry Billingsley

An “Intelligent Book” is a Web-based textbook that contains exercises that are backed by computer models or reasoning systems. Within the exercises, students work using appropriate graphical notations and diagrams for the subject matter, and comments and feedback from the book are related into the content model of the book. The content model can be extended by its readers. This dissertation examines the question of how to provide an Intelligent Book that can support undergraduate questions in Number Theory, and particularly questions that allow the student to write a proof as the answer. Number Theory questions pose a challenge not only because the student is working on an unfamiliar topic in an unfamiliar syntax, but also because there is no straightforward procedure for how to prove an arbitrary Number Theory problem.

The main contribution is a system for supporting student-written proof exercises, backed by the Isabelle/HOL automated proof assistant and a set of teaching scripts. Students write proofs using MathsTiles: a graphical notation consisting of composable tiles, each of which can contain an arbitrary piece of mathematics or logic written by the teacher. These tiles resemble parts of the proof as it might be written on paper, and are translated into Isabelle/HOL’s Isar syntax on the server. Unlike traditional syntax-directed editors, MathsTiles allow students to freely sketch out parts of an answer and do not constrain the order in which an answer is written. They also allow details of the language to change between or even during questions.

A number of smaller contributions are also presented. By using the dynamic nature of MathsTiles, a type of proof exercise is developed where the student must search for the statements he or she wishes to use. This allows questions to be supported by informal modelling, making them much easier to write, but still ensures that the interface does not act as a prop for the answer. The concept of searching for statements is extended to develop *massively multiple choice* questions: a mid-point between the multiple choice and short answer formats. The question architecture that is presented is applicable across different notational forms and different answer analysis techniques. The content architecture uses an informal ontology that enables students and untrained users to add and adapt content within the book, including adding their own chapters, while ensuring the content can also be referred to by the models and systems that advise students during exercises.

Acknowledgements

There are many people without whom my research would have been much more difficult. I thank my supervisor, Professor Peter Robinson, for the advice he provided throughout this PhD. This dissertation formed part of a wider research project involving researchers from Cambridge and the Massachusetts Institute of Technology. Discussing ideas and material with those researchers – Hal Abelson, Gerry Sussman, Chris Hanson, Mark Ashdown, and Kasim Rehman – has been valuable and informative. Alan Blackwell provided advice on how to run a usability study, and Sparsh Gupta assisted me in conducting the usability study in Chapter 9. Thanks are also due to Joe Hurd, Michael Compton, and Larry Paulson for their help as I learned to use Isabelle/HOL.

Tal Sobol-Shikler, Mark Ashdown, and Shazia Afzal kindly read and reviewed early drafts of chapters. I also thank the referees and editors of the *Journal of Automated Reasoning* – their extensive feedback and corrections on one of my papers has also helped me to improve large sections of this dissertation. I also thank my examiners, Mateja Jamnik and Robert Harding, for their helpful corrections and for the interest they showed in my research.

My research was funded and supported by the Cambridge-MIT Institute and the Cambridge Commonwealth Trust. My parents, John and Ros, also provided financial and moral support, especially in the first few months of my research.

Finally, I thank my wife Fiona. She has been endlessly patient and supportive, and has moved half way around the world with me so that I could undertake this PhD. Without her understanding and companionship, it might have been too difficult to contemplate.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

This dissertation is not an Intelligent Book. It uses the same words to say the same thing to every reader regardless of whether or not they can understand it. It cannot help readers to work through example problems and it cannot say anything that is not already in the book. In many situations, a static unintelligent book like this is appropriate. This thesis has to be examined and that would be much harder to do if it changed every time it was read. However, if a textbook is going to be presented on a networked computer, then sometimes it makes sense to take advantage of the capabilities that the computer and the network can provide. This dissertation examines how we can build an Intelligent Book that can support students learning introductory Number Theory, but designed using techniques that are applicable to any scientific or mathematical subject.

1.1 Background

1.1.1 "The Industrial Revolution in Education"

In 1926, long before electronic computers became available, Sidney Pressey built machines that could ask students multiple choice questions [Pre26]. Photographs of a Pressey Testing Machine are shown in Figure 1.1. The machine showed the number of the question on a counter. Students would read the text of the corresponding question from a card that also listed the possible answers. They would then push one of the five buttons on the machine to enter their answer, and pull a lever to move on to the next question. Depending on how it was configured, the machine would either tally the answer as right or wrong and move on, or would refuse to move to the next question until the correct answer was entered. The correct answers were held on a roll of punched paper inside the machine, similar to a pianola reel.

Psychologists in the twentieth century, including B.F. Skinner [Ski54, Ski58], hoped that mechanisation could bring the same kind of revolution to education that it had to industry. Machines would automate as many of the mundane parts of teaching as possible so that human teachers could spend more time on the parts that require their expertise. So, for example, a Pressey machine would enable students to receive feedback on as many questions as they

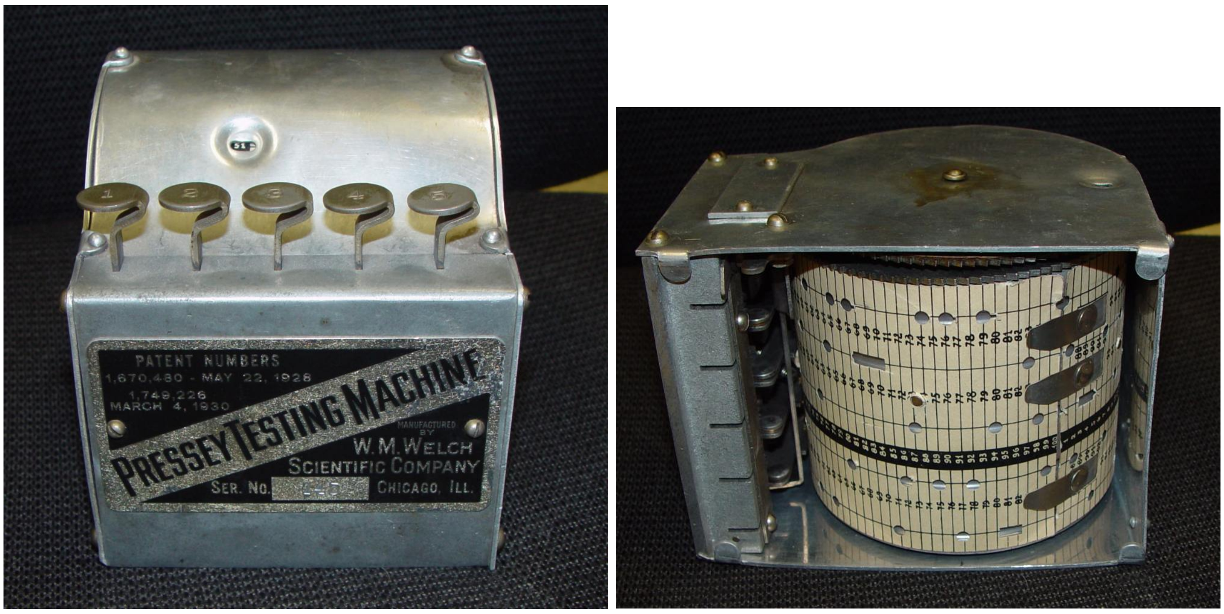


Figure 1.1: External and internal views of a Pressey Testing Machine.

like at their own convenience, without troubling a human marker. Pressey only sold 120 of his machines, but arguably the revolution in education did take place, at least in assessment. Computer-marked multiple choice exams are now a commonly used examination technique both at school and at university. The US Graduate Record Examination (GRE) General Test, an admission requirement to many graduate schools, is a computer-based test.

1.1.2 Bloom and the Two Sigma Problem

In 1984, educational researcher Benjamin Bloom published his “Two Sigma Problem” paper [Blo84] that is one of the most cited papers by educational technologists. The paper starts with a result observed by two of his doctoral students [Ana83, Bur84], that individually tutored US high school students performed two standard deviations (“two sigma”) better than students taught in classes of thirty. This means that the average tutored student performed better on tests than 98% of the classroom taught students. (Cohen, Kulik and Kulik [CKK82] also confirmed that small group tutoring outperforms classroom teaching, although in their study the margin was smaller.)

Educational technologists often cite only this result from Bloom’s paper. It is a motivation for providing more personal attention to students’ individual needs, and for examining the pedagogical techniques that human tutors use and trying to replicate them in automated systems. Bloom’s paper itself goes on to examine strategies, and combinations of strategies, that classroom teachers can use to bridge the two sigma gap. Table 1.1 shows a selection of strategies and the learning gains Bloom found they produced. Many of these are clearly applicable to educational technology, and whether intentionally or not, most automated teaching systems include one or more of these strategies. For example, simply grading homework questions is found to improve learning by 0.8σ, and Mastery Learning (re-teaching items that were not understood) gave a 1σ improvement. Unsurprisingly, then, automated teaching systems have also been shown to produce learning gains in students when compared to classroom teaching alone [SP96, SST+01, MMSM01].

Strategy	Effect size	Percentile Equivalent
Tutorial instruction	2.0σ	98
Enhanced Cues and Participation (better explanations and more student participation)	1.5σ	93
Reinforcement (rewarding desirable behaviours, eg praising a student who gives a correct response in a discussion)	1.2σ	88
Increasing students' time on task	1.0σ	83
Mastery Learning (re-teaching items that most students did not grasp)	1.0σ	83
Assigning and grading homework	0.8σ	79
Enhanced Pre-requisites (ensuring pre-requisite material is understood)	0.6σ	73
Assigning homework	0.3σ	62
Asking higher order questions	0.3σ	62
Combination	**Effect size**	**Percentile Equivalent**
Enhanced Cues and Participation + Reinforcement + Mastery Learning	1.7σ	96
Enhanced Pre-requisites + Mastery Learning	1.6σ	95

Table 1.1: A selection of active teaching techniques that can improve student performance. Extracted and adapted from Bloom [Blo84], in turn using data from Walberg [Wal84], Burke [Bur84], Anania [Ana83], Leyton [Ley83], and Tenenbaum [Ten82].

1.1.3 Recent Research

Recent research has continued trying to reduce the cost of education and improve its outcome. As more materials have become Web-enabled, it has also looked at ways of providing individual teaching to remote students. Many of these systems are described in the Related Work in Chapter 2. *Intelligent Tutoring Systems* have been designed to apply pedagogical techniques, based on either theories of cognition or observations of human tutors, to teach many different subjects. *Intelligent Learning Environments* have considered how exercises and content fit within a course, and can generate tailored lessons for individual students.

1.1.4 Complementing the Tutor

My research has been conducted at the University of Cambridge, and it is worth taking a moment to consider the local teaching situation. The University provides small group tutorials, called "supervisions", to its undergraduate students in each of their lecture courses. This is an approximation to Bloom's ideal of individual tutoring by an expert tutor. So, automated teaching systems would be unlikely to produce the same learning gains in Cambridge that they have been shown to produced in untutored students. However, this does not mean that automated teaching systems have no role to play.

In the Computer Laboratory, approximately one hour of tutorial is given for every four hours of lectures, in groups of no more than three students. For a 16 lecture course with 120 students, at least 160 hours of tutorials take place in total. This does not take into account preparation time or the time taken to mark students' homework. This is very labour intensive, making it difficult to provide more tutorials even if they could improve results. Not only is cost an issue, but it can be hard to find enough suitable tutors and time during the term for the tutorials to take place. Furthermore, because many tutors are graduate students with little or no formal training in tutoring, there is some variation in their teaching skills.

An Intelligent Book, as an automated teaching system, could serve two useful purposes in this setting. Firstly, there are often common homework problems that tutors set and common misconceptions that students tend to have. If students become stuck on a homework problem without automated assistance, then this cannot be resolved until the following tutorial, and working through the rest of the question takes up valuable tutorial time. An Intelligent Book could target these examples and misconceptions, allowing tutors to dedicate more time to the students' less common needs. Secondly, by being a common resource available to all students, an Intelligent Book could help even out the quality of tutoring that each student receives.

1.1.5 Why a Textbook

At some point, an automated homework system has to be able to correct students about factual errors. This involves describing a piece of content, so it is useful if the exercise can be combined with some kind of content system. The conventional take-home resource that students use as a source of exercises and content is a textbook.

The role of a textbook affects the way students interact with it, and this is important to preserve. A textbook is always the students' servant, never their master. It does not nag students about when a piece of coursework is due. It does not mark their work for summative assessment, so students are free to get exercises wrong without penalty. The model in this dissertation, then, is for an automated system to take the role of the textbook, rather than the tutor (as in *Intelligent Tutoring Systems*) or the course structure (as in *Courseware Management Systems* and many *Intelligent Learning Environments*).

1.2 This Dissertation

This dissertation seeks to develop a Web-based Intelligent Book that can support proof exercises in introductory Number Theory. There are two parts to this: developing technology to support an Intelligent Book, and developing proof exercises within the Book. The second part is the more challenging.

1.2.1 Challenges for an Intelligent Book

An Intelligent Book should be able to cover all the topics within a course. This could involve a wide variety of graphical notations, styles of interaction, and content. For example, a book for electronic circuits may need to include exercises working with circuit diagrams, simulation plots, digital timing diagrams, and potentially various kinds of engineering plot. Consequently, the architecture for an intelligent book should be able to support different graphical notations and different modelling or reasoning systems to support those notations.

The content of an Intelligent Book should be extensible and adaptable. It should be possible for both the teacher and students to add new material or improved versions of material into the book. Also, when students work through a subject they do not always rely on a single explanation for each item. Reading lists for courses often recommend more than one textbook, and students might often use additional material from the Internet. An Intelligent Book should support the concept that often there is not a definitive explanation that is suitable for all students, and that having alternative explanations of the same material can be helpful. At the same time, however, the automated help and advice that the system gives to students must be able to refer to the content.

The appropriate pedagogies and the depth of analysis to use can also differ from question. For some questions, we can model students' solution steps exactly and train them in a particular procedure. However, other questions may involve problem solving or design tasks where there is no known step-by-step procedure that the system can assess students against. In Bloom's Taxonomy of Educational Objectives [Blo56], this means moving from the lowest *knowledge* level (that includes practising taught techniques) to the *application* and *synthesis* levels, where students must work out for themselves how to design a solution.

1.2.2 Challenges for the Proof Exercises

Number Theory proofs are an example of a difficult domain where there is no known step-by-step procedure that can complete an arbitrary proof. The automated systems that have been built for proofs need a great deal of guidance from their users to prove most theorems. A teaching system for proof faces the challenge of helping students who do not know how to complete a proof using a reasoning system that cannot complete the proof either.

Automated proof assistants are also known to be difficult to learn: it can often be harder to write a verifiable proof in a proof assistant than it is to prove the theorem manually on paper. Enabling students to write automatically verifiable proofs, and making the system's reasoning understandable to students are both significant challenges.

Mathematics is a difficult language to work with over the Web. Keyboards are designed for a one dimensional syntax (text) whereas mathematics is often two dimensional and includes layout. The terminology used by mathematical modelling systems can also be very specific and difficult for novices to learn. Students working with an automated system for mathematics therefore face the difficulty of working on an unfamiliar subject using an unfamiliar notation.

1.2.3 Outline of the Following Chapters

Chapter 2 describes previous work by other researchers that is relevant to this dissertation. Chapters 3 to 5 describe how the architecture of the Intelligent Book supports complex questions that can include different graphical notations, different teaching pedagogies, and different modelling or reasoning systems:

- Chapter 3 introduces these complex graphical questions and describes how the client components are organised.
- Chapter 4 describes the content model of the Book, that allows students to add and alter content while still allowing the automated teaching advice to refer to it.
- Chapter 5 describes the structure of the teaching scripts, and how they allow different pedagogies and different modelling or reasoning systems to be used.

Chapters 6 to 9 describe the formally modelled proof exercises:

- Chapter 6 provides the background, describing the usability issues with automated proof systems and developing specific design goals for the proof exercises.
- Chapter 7 introduces MathsTiles, a simple structured interaction language I developed for mathematics.
- Chapter 8 describes how MathsTiles is used as a language for writing automatically verifiable proofs.
- Chapter 9 describes an evaluation and usability study I conducted of the proof exercises.

Chapter 10 uses the results of the usability study, and separate observations of students answering proof questions in front of human tutors, to develop informally modelled proof questions. These rely on the fact that the teacher already knows the arguments students are likely to make to simplify the modelling and make questions much simpler to write. The informal modelling principle is extended to develop *massively multiple choice* questions.

Finally, Chapter 11 concludes the dissertation.

1.2.4 Publications

Some of the work described in this dissertation has appeared in the following publications:

1. William Billingsley and Peter Robinson. Searching questions, informal modelling, and massively multiple choice. *International Conference of the Association for Learning Technology (ALT-C)*, 2007. in press.
2. William Billingsley and Peter Robinson. Student proof exercises using MathsTiles and Isabelle/HOL in an Intelligent Book. *Journal of Automated Reasoning*, 2007. in press.
3. Kasim Rehman, William Billingsley, and Peter Robinson. Writing questions for an Intelligent Book using external AI. In *Proceedings of the Sixth International Conference on Advanced Learning Technologies (ICALT2006)*, pages 1089 – 1091, 2006.
4. William Billingsley and Peter Robinson. Towards an intelligent textbook for discrete mathematics. In *Proceedings of the 2005 International Conference on Active Media Technology, Takamatsu, Japan*, pages 291 – 296, 2005.
5. William Billingsley, Peter Robinson, Mark Ashdown, and Chris Hanson. Intelligent tutoring and supervised problem solving in the browser. In *Proceedings of the IADIS International Conference WWW/Internet 2004, Madrid, Spain*, pages 806 – 811, 2004.
6. William Billingsley and John Billingsley. The animation of simulations and tutorial clients for online teaching. In *Proceedings of the 15th Annual Conference for the Australasian Association for Engineering Education and the 10th Australasian Women in Engineering Forum, Toowoomba, Australia*, pages 532 – 540, 2004.

CHAPTER 2

Related Work

My research in this dissertation touches on previous work in a number of areas. A wide variety of automated homework systems exist that support questions in different subjects, both on-line and as stand-alone programs. Intelligent Tutoring Systems research has examined appropriate pedagogies and teaching methods for an automated question system. Intelligent Learning Environments and Adaptive Hypermedia research has examined how content material can be adapted to the needs of individual students. Other research projects have looked at how to edit mathematics and other structured languages, and there are also a number of educational systems that have been built to teach logic and proof. For simplicity's sake, each system described in this chapter is listed under only one heading, although there is some overlap between the sections.

2.1 Automated Questions

2.1.1 Short answer and multiple choice systems

UWA-CPCS [Sco96] is a Hypercard-based question system developed at the University of Western Australia in the mid-1990s. It supports questions where the answer is a number, and uses an eighty-twenty rule to provide useful feedback very simply. Roughly 80% of student mistakes on a question fall into a small set of "common errors". When one of these common wrong answers is encountered, UWA-CPCS shows a pre-written Hypercard explaining the mistake that leads to that wrong answer. An updated version, called JellyFish / FlyingFish [SS98], uses the same technique but allows Java applets to replace the Hypercards. For example one applet (Figure 2.1) shows a picture of a tissue sample, and asks the student to move labels over particular kinds of tissue. The applets can either handle the marking of the answer themselves, or send a short answer (eg, the co-ordinates of the tissue labels) to the server.

Alice Interactive Mathematics (AIM) [KKVdB00] is a short answer system that solves the problem of mathematically equivalent answers by connecting to the Maple mathematics system at the server – answers that Maple considers to be equivalent are deemed correct. To prevent the student from taking unfair advantage of this, AIM maintains a list of forbidden words for a question. For example in a question asking a student to calculate $sin(\pi/2)$, then sin would be

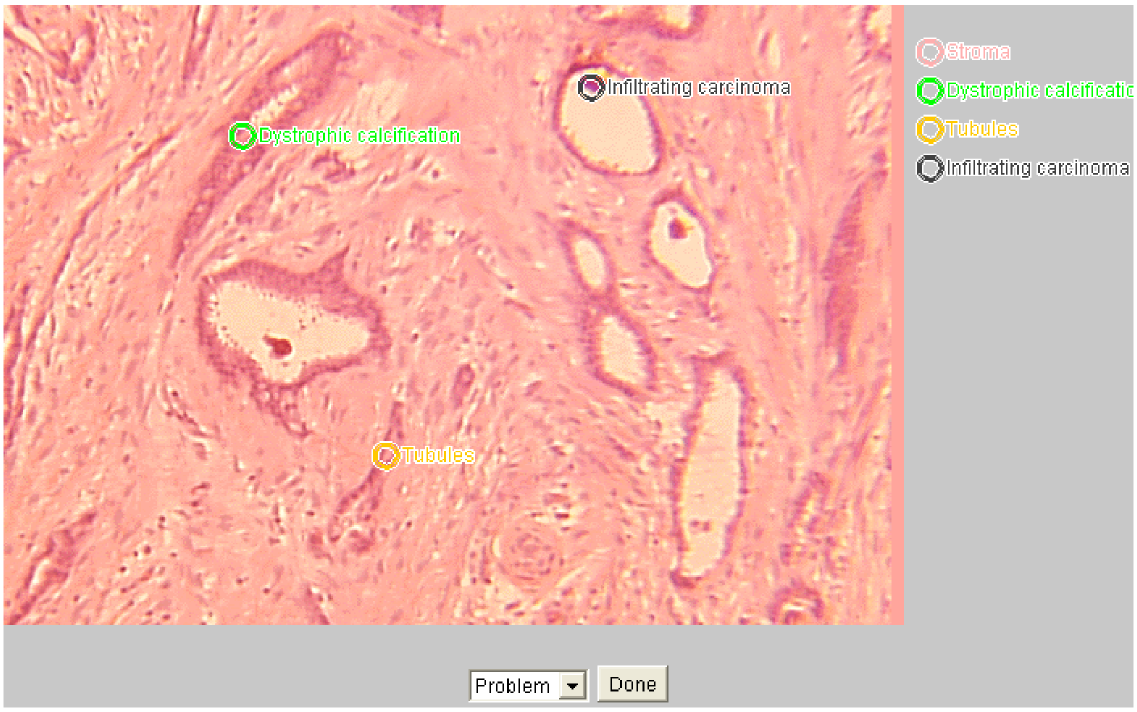


Figure 2.1: A tissue identification question in JellyFish.

made a forbidden word to prevent from entering $sin(\pi/2)$ as the answer. AIM also provides a hint system, whereby the student can click a button to buy a hint for a small point penalty. The hint can be pre-written text, or it can be a sub-question that the student will need to answer on the way to answering the complete question.

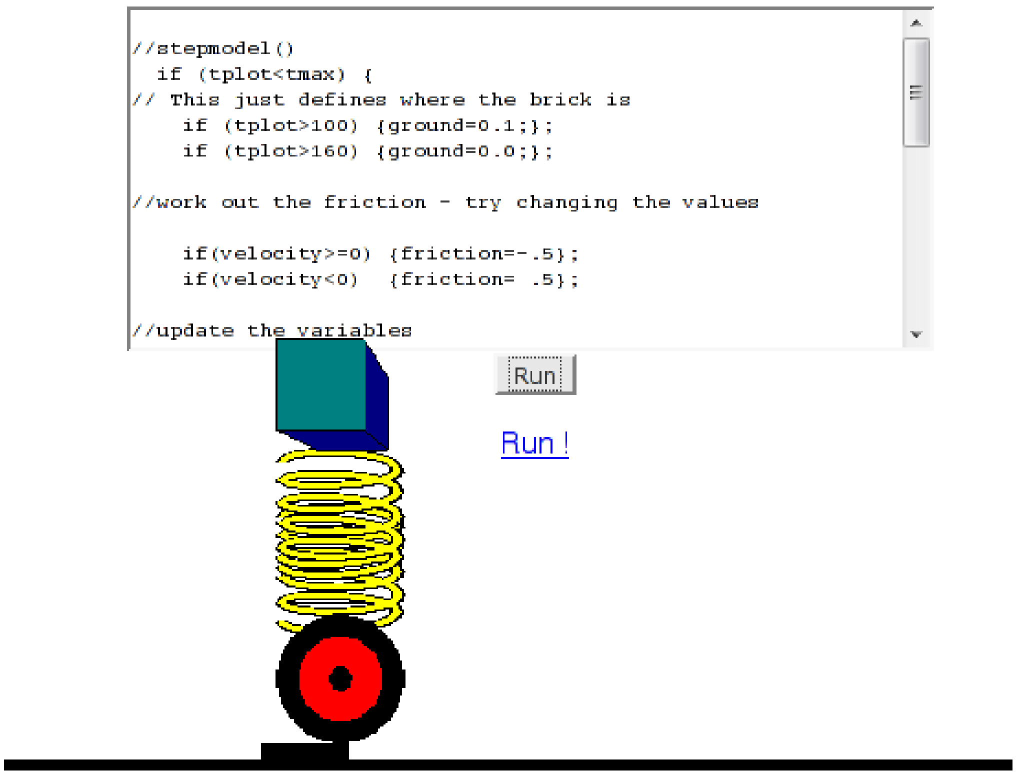
The MIT 6.001 tutor [LP00] is a Web based short answer system that is used in MIT's courses on artificial intelligence and the Scheme programming language. The tutor supports questions where the student is asked to answer by writing a Scheme routine. A checking function for the question runs the student's routine against a set of test cases in order to see whether the routine does what was asked for.

SIETTE [GRC02, CGM$^+$04] is a Web based system for *User Adaptive Tests*. These are tests where the next question is chosen based upon the student's performance in the test so far; they are useful because they allow a more accurate gauge of a student's skills and knowledge using fewer questions than fixed tests. Each question is assigned a *category*, which designates what skill or knowledge item the question assesses, and an *Item Characteristic Curve*, which broadly speaking is a curve plotting a hypothetical student's "knowledge level" against the expected probability that the student would get the answer right. When SIETTE is satisfied that it has an accurate gauge of the student's knowledge in a given category, it will stop asking questions for that category. SIETTE questions can either be multiple choice items, or can use a small applet, rather like JellyFish. Examples of SIETTE applets include putting a set of five pictures of buildings in order of their construction date, and painting the region of a map where a particular species of tree grows.

2.1.2 Online simulation

Science Learning Spaces [KSF99] was a project looking to develop rich learning environments based around the availability of large numbers of online simulations. Students would be able to explore information and try out the various simulations, and construct diagrams that represent their conceptual module of the material. The project developed a "feasibility demonstration" that combined the Active Illustrations [For97] simulation framework with Belvedere [SCL$^+$01, Sut03], a coached environment for drawing graphical representations of an argument or conceptual model. A plug-in architecture for intelligent tutors [RK97] was also included to train students how to conduct an experiment.

JOLLIES [Bil01, BB04] are a set of Web-based simulations developed at the University of Southern Queensland for use in engineering and mechatronics courses. The simulations are written in Javascript, and animated on the Web page using calls into the browser's Java plugin. More importantly, however, the part of the program that represents the simulation (as opposed to the animation code) is exposed to the student in an edit box. Students are expected to alter this code and see how their changes affect the behaviour of the system. Exercise notes with each simulation provide a guided set of changes the student ought to examine. These usually include altering gains, friction values, and the size of the time step. The intent of JOLLIES is both to improve students' understanding of the system, and also to improve their understanding of how to model an engineering system. A screenshot of a JOLLIES simulation is shown in Figure 2.2.



```
//stepmodel()
  if (tplot<tmax) {
// This just defines where the brick is
    if (tplot>100) {ground=0.1;};
    if (tplot>160) {ground=0.0;};

//work out the friction - try changing the values

    if(velocity>=0) {friction=-.5};
    if(velocity<0)  {friction= .5};

//update the variables
```

Figure 2.2: A JOLLIES simulation for teaching Control Theory. Part of the code that describes the simulation is exposed so that the student can alter it and see how those alterations affect the behaviour of the system.

A number of projects have developed online simulations of experiments that students might traditionally conduct in a laboratory. Jade [AT00] supports students learning VLSI design by allowing them to design circuits and then examine their behaviour by attaching virtual probes to the circuit diagram and examining the probes' output traces. Roberts's Virtual Machines Laboratory [Rob04] supports Web-based simulations of a transformer, a synchronous machine, and an induction machine. RIDES [MJP$^+$97] is an authoring environment for simulation based tutorials. It was originally delivered over X, but a Java version for intranets was later developed. RIDES provided a toolkit for placing graphical objects on the screen and attaching programmatic behaviour to them. (This part of the system is somewhat similar to Adobe's commercial Flash toolkit.) Procedural instructions can then be added – these tell the student what actions they should perform in the simulation; if the student carries out the actions incorrectly, RIDES can correct them and record performance measures.

More recently, virtual simulations have been extended with the idea of teleoperation, to allow students to conduct real experiments remotely. Jochheim and Röhrig's Virtual Lab [JR99] allows students to control a four wheeled vehicle remotely, and provides the software infrastructure for other experiments. "WebLabs" and "iLabs" have been developed to allow students to teleoperate experiments in microelectronics [HdAC$^+$04] and chemical engineering [SGK$^+$05, SKCM06].

2.1.3 Summary

The eighty-twenty rule that most students tend to make the same sort of mistake is a useful observation that is used both in my work and also in the *buggy rules* of Cognitive Tutors (described in the next section). The systems described here also show that there can be a wide variety of useful analysis techniques – from asking an external system to assess the answer (in the case of AIM) to executing the student's answer against a series of test cases. The simulation examples also show that a system can be educationally useful even if there is no deliberate tutorial feedback: the student gains experience from working with the simulation and can see the consequences of his or her mistakes.

2.2 Intelligent Tutoring Systems

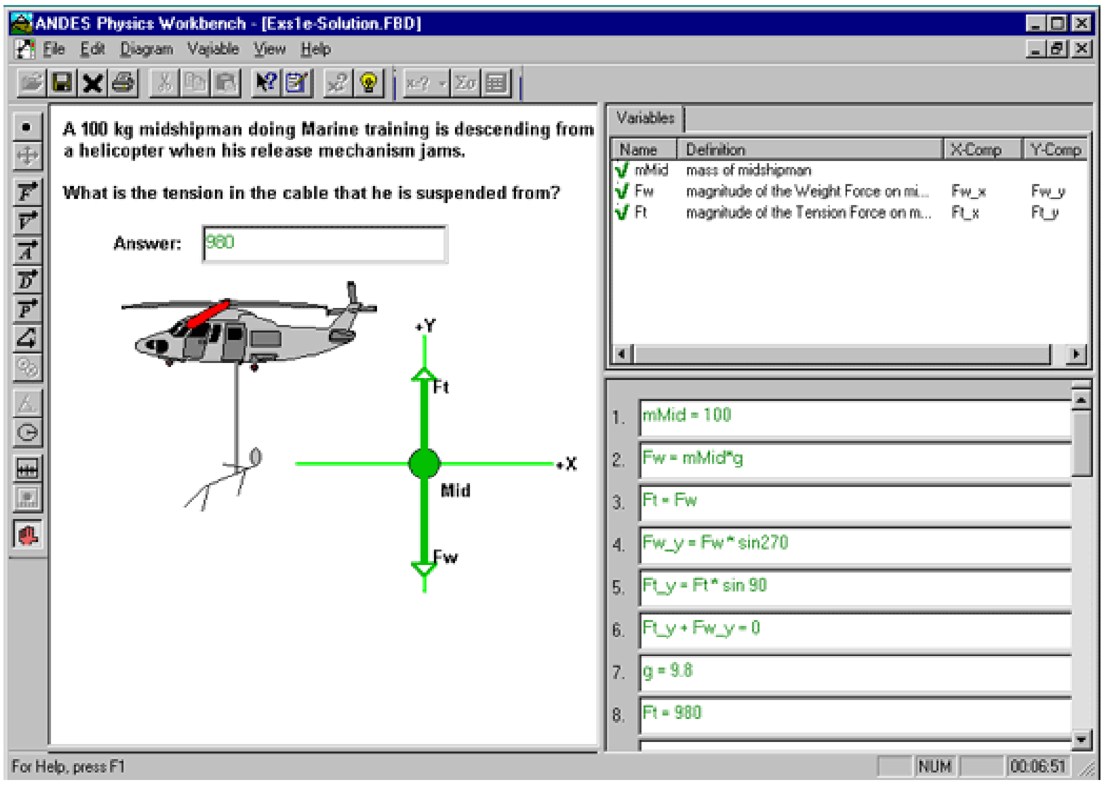
Intelligent Tutoring Systems (ITS) research is heavily motivated by Bloom's "two sigma" finding – that tutoring students individually is so much more effective than applying the same classroom-wide teaching to all students. ITS research attempts to automate the pedagogy of a human tutor. The "Intelligent" part of Intelligent Tutoring Systems, then, refers to the implementation of the pedagogy, while the marking of answers to individual questions can often be very simple. I have grouped the systems in this section according to the pedagogical and cognitive theories that they are built upon.

2.2.1 Model Tracing

Anderson's ACT [And83] and ACT-R [And93, And96] theories of cognition separate knowledge into two kinds: *declarative* and *procedural*. Declarative knowledge includes facts and theoretical laws. Procedural knowledge describes what action to take in a given situation, and is represented as a set of production rules. The Cognitive Tutors from Carnegie Mellon University [ACKP95], based on ACT and ACT-R, introduced Model Tracing as an automated tutoring technique. They hold a set of declarative and production rules describing the process that a "good student" would follow to answer the question. They can also hold *buggy rules* that represent common misconceptions. The tutor attempts to interpret (or *trace*) students' actions by comparing them with its cognitive model of how a good student should behave. Each action is evaluated to see whether any combination of production rules could have produced it. If the action does not follow from the production rules, or if it matches a buggy rule, then an error is flagged and the tutor gives the student corrective feedback. Effectively, the tutor trains students to behave like a model student.

A Bayesian network is often kept to calculate the probability that each student understands each of the production rules. This is constantly updated, in a process called *knowledge tracing*, and a summary of it is often shown to the student as a skill-meter.

The Carnegie Mellon research group has developed Cognitive Tutors for high school algebra [KAHM97, Rit97], high school geometry [ABY85], and programming in LISP [CT00, CB97]. Andes [GV00, SST$^+$01, CGV02, VLS$^+$05] is a particularly successful Model Tracing tutor used to teach Newtonian physics at the US Naval Academy. Typical questions involve calculating the forces acting in particular systems – for example, if a 100kg marine is suspended by a cable underneath a helicopter, what forces are acting upon the cable. Students are asked to draw diagrams representing the forces involved, describe the equations that relate each of the forces, and then solve the equations. An example of an Andes question is shown in Figure 2.3.


ANDES Physics Workbench - [Exs1e-Solution.FBD]
File Edit Diagram Variable View Help
+Y
Ft
+X
Mid
Fw
For Help, press F1
NUM
00:06:51

A 100 kg midshipman doing Marine training is descending from a helicopter when his release mechanism jams.

What is the tension in the cable that he is suspended from?

Answer: 980

Variables

Name	Definition	X-Comp	Y-Comp
✓ mMid	mass of midshipman		
✓ Fw	magnitude of the Weight Force on mi...	Fw_x	Fw_y
✓ Ft	magnitude of the Tension Force on m...	Ft_x	Ft_y

1. mMid = 100
2. Fw = mMid*g
3. Ft = Fw
4. Fw_y = Fw * sin270
5. Ft_y = Ft * sin 90
6. Ft_y + Fw_y = 0
7. g = 9.8
8. Ft = 980

Figure 2.3: A Newtonian physics question in Andes.

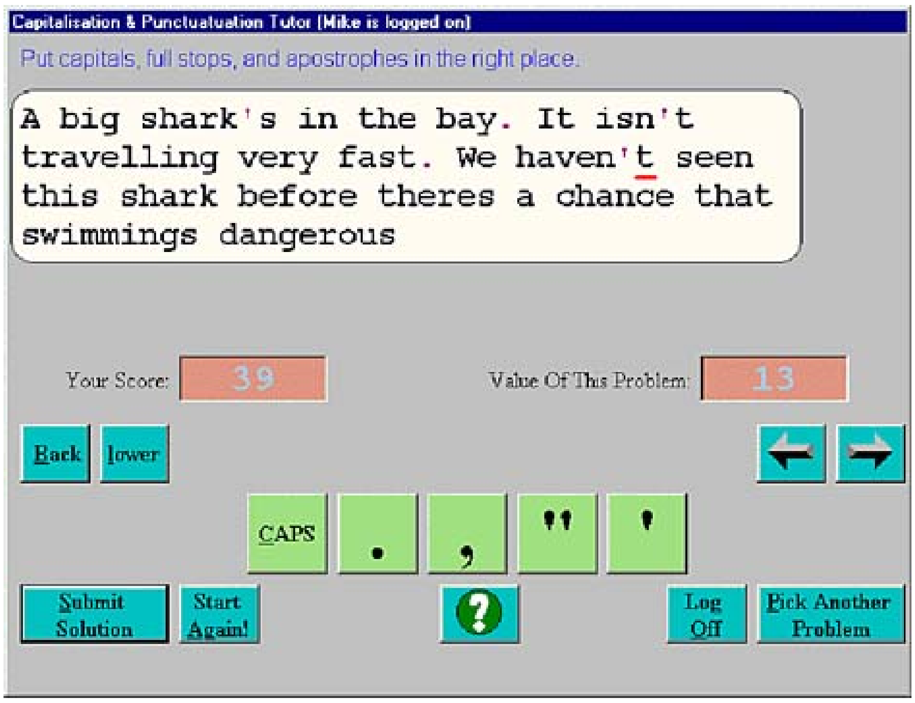
Identifying and programming the production rules for a Model Tracing tutor can be time-consuming. Some recent work [KAH$^+$04, AMSK06] has focussed on authoring tutors by example. Rather than immediately develop a full set of cognitive rules for the question, the system is shown a number of examples of correct and incorrect solutions. The steps in the examples are recorded into a *behaviour graph*. This behaviour graph can then be used directly as the model in an *Example Tracing Tutor*, or it can be used as an aid to develop the production rules for a Model Tracing tutor.

2.2.2 Constraint Based Tutoring

Constraint Based Tutoring (CBT) is an approach proposed by Ohlson [Ohl92]. Students' mental processes are too complex for any system to model completely and accurately, so tutoring systems must rely on a model that is incomplete but useful. The approach CBT takes is to model knowedge as a set of constraints on answers in a domain. Correct solutions are those that do not violate any constraints. Each constraint has a relevance condition and a satisfaction condition. If the relevance condition is met, then the satisfaction condition must also be met or the answer has violated the constraint.

Generally, the constraints represent the fundamental rules of the domain: for example, the laws of physics or the rules of punctuation. CBT assumes that no good solution to a question can traverse a problem state where one of the fundamental principles is breached. So, students are free to take any actions they like, so long as they do not cause their answers to enter an invalid state [MKM03]. A Bayesian Network is usually kept to estimate the students' understanding of each rule.

Constraint Based Tutors have been developed for the SQL database language [MO99, MH00], entity relationship modelling in databases [SM02], data normalisation [Mit02], and English language punctuation [MM01]. A screenshot of a CAPIT punctuation question is shown in Figure 2.4.


Capitalisation & Punctuatuation Tutor [Mike is logged on]

Put capitals, full stops, and apostrophes in the right place.

A big shark's in the bay. It isn't travelling very fast. We haven't seen this shark before theres a chance that swimmings dangerous

Your Score: 39

Value Of This Problem: 13

Figure 2.4: A punctuation question in CAPIT

2.2.3 Mixed-Initiative and Conversational tutors

For both CBT and Model Tracing, the tutoring session is primarily led by the system. CBT sets the student questions and corrects broken rules; Model Tracing encourages the student to follow an expected procedure to reach the answer. Mixed-Initiative tutors instead support the principal that both the system should be able to ask the student questions, and also the other way around. Tutoring becomes a two-way conversation between the student and the tutor. Even if the tutor does not consciously adapt its behaviour to an individual student, the fact that each student will make different actions or say different things will cause the sessions to progress in unique ways.

The earliest Mixed-Initiative system is usually regarded to be Carbonell's SCHOLAR [Car70] system for teaching South American Geography. It holds a network of facts, concepts, and procedures as a database, and when it identifies a student misconception it tries to show materials that will help the student to see her own error. SCHOLAR has two modes: a tutor led mode in which SCHOLAR asks the student questions, and a student led mode which works the other way around. Its dialogue however does not support a coherent conversation, in that each new question can be very disconnected from the last.

SOPHIE [BBB75] is designed to be a *Reactive Learning Environment*: the student is encouraged to try out his ideas and receive detailed feedback based on a computer model of the scenario. In a SOPHIE scenario, the student is given an electronic circuit that has a fault in it, perhaps a damaged component or a short-circuit, and is asked to diagnose the fault. He can ask SOPHIE to make measurements on the circuit, and can propose hypotheses about what the fault is. If a hypothesis is logically flawed, then SOPHIE describes how it does not match the measurements that have been taken so far. If the student becomes stuck, SOPHIE can generate plausible hypotheses for him to explore (again based on the measurements so far). The computer model also allows the student to ask SOPHIE hypothetical questions such as "what would the output voltage be if the beta of Q3 were 8", or "what would happen if R22 were shorted?". An extract from a SOPHIE tutorial session is shown in Figure 2.5.

Another approach has been to examine the dialogue that takes place in human tutorials, and attempt to reproduce this conversational style of teaching in an automated system. The AutoTutor project [GWHWH$^+$99, CHG$^+$00, GLJ$^+$04] pioneered this approach. AutoTutor is a natural language dialogue tutor for Newtonian physics and computer literacy. It asks questions that require a conceptual explanation as the answer. For example, "*When a car without headrests is struck from behind, the passengers often suffer neck injuries. Why do passengers get neck injuries in this situation?*". A student's initial responses is often very short – just one or two words. The tutorial dialogue then focusses on asking the student to explain and elaborate, in order to draw out more of the learner's knowledge. The conversation is mixed-initiative in that both the tutor and the student can ask questions and start new topics of discussion. Other notable conversational tutors include Atlas-Andes [RJR$^+$], which adds the Atlas conversational module to the Andes physics tutor described earlier, and CIRCSIM-Tutor [LEM04], which examined how an automated tutor can use appropriate analogies to help explain concepts to students.

2.2.4 Summary

There is no one-size-fits-all pedagogy for automated tutoring. Model Tracing tutors are carefully designed to teach students how to execute a known process, but this also means they are only suitable for questions where the process used to answer a question is well-defined. For example, Model Tracing does not easily apply to larger design tasks that may involve exper-

```
Student:  REPLACE Q6
Sophie:   I am going to ask you some questions about how Q6 is
          faulted.  Are there any junctions shorted?
Student:  I DON'T KNOW
Sophie:   Are any terminals open?
Student:  YES
Sophie:   Which terminals are open?
Student:  THE BASE
Sophie:   Q6 is not faulted in that manner.  Q6 not replaced.
Student:  IS IT POSSIBLE THAT Q6 HAS AN OPEN BASE
Sophie:   Is the complete fault mode EB/EC/BC = OP/OK/OP what
          you intended?
Student:  SURE
Sophie:   When the settings were:  CC=1.0, CR=HIGH, LOAD=20,
          VC=1.0, VR=HIGH, that fault conflicts with the
          following measurement.  You observed that the
          output voltage was 11.7 volts.  If Q6 had fault mode
          EB/EC/BC = OP/OK/OP it would be 23.9 volts.  In a
          working circuit it is 19.9 volts.
```

Figure 2.5: An extract from a SOPHIE session

imenting and prototyping in order to develop a good answer, and where there are an almost infinite number of choices of next step. Reactive Learning Environments would be more appropriate for those questions. The natural language dialogue approach is useful to help students understand the core concepts and issues in a domain, but natural language can be too vague for questions involving detailed mathematical equations.

Constraint-Based Tutoring assumes that no correct answer can traverse an invalid state, which is not strictly true for some design tasks. For example, engineers are sometimes encouraged to take a "rough cut" approach first, deliberately ignoring some rules in order to get an approximate answer that can then be refined.

2.3 Web-based Learning Environments and Adaptive Hypermedia

2.3.1 AlgeBrain

AlgeBrain [ASF99] is a Model Tracing tutor that was converted into a Java Applet for use over the Web. It teaches elementary high school algebra, particularly how to solve algebraic equations. However, AlgeBrain also includes a *Just-In-Time Dictionary* containing some content material. For example, if students left-click on the "`Collect like terms`" task button, they indicate that is the next step they are taking in their solution. However, if they right-click on the button, a dictionary entry is shown explaining what it means to collect like terms in an equation. This allows students to see explanations of content material in context as they become appropriate for the problem at hand, rather than in isolation. The combination of teaching material with tutorial advice gives AlgeBrain some of the features of a Web-based Learning Environment.

2.3.2 ELM-ART

ELM-ART [BRW96, WB01] is described as an adaptive electronic textbook for programming in LISP. It contains lessons which are divided hierarchically into sections, subsections, and units. It also contains “live examples” (underlined LISP expressions that can be run in an interactive LISP evaluator by clicking on them) and short programming problems.

ELM-ART keeps a four layered user model for each unit: has the user visited the unit, which test items has the user attempted and were they successful, can the unit be inferred as known from another unit, and has the unit been manually marked as known. This model is used to annotate links within the book. For example, green balls are shown next to links that ELM-ART recommends the student should visit next, while red balls are shown next to links that ELM-ART does not think the student is ready for yet. There is also a “`Next Topic`” button that asks ELM-ART for the best next step depending on the knowledge state of the learner.

2.3.3 REDEEM

REDEEM [AMG$^+$03, AG04] is a system designed to enable non-technical teachers to reuse existing computer-based instructional material, for example Web pages, within an intelligent teaching system. The teachers add metadata that describes each page in terms of a number of important dimensions, such as its *difficulty* and its *familiarity*. Simple kinds of questions (for example, *true/false* and *multiple choice*) can also be added, along with hints. During operation, an Intelligent Tutoring Shell models the individual students and selects appropriate parts of the course to present to them. The instructional strategy that REDEEM uses can be configured using a graphical interface.

REDEEM is perhaps the most extensively evaluated learning environment that has been developed, and has been shown to give improved learning outcomes with students compared to “dumb” courseware. However, the exact causes of the learning gains are harder to identify, and Ainsworth (the principle developer) suggests they might simply be due to increasing the amount of time students spend on a task and providing feedback on their errors [Ain06].

2.3.4 The Living Book

The Living Book [BGHS02, BFGHS04] project has developed an online adaptive book for teaching logic to computer scientists. Its content model is based on *Slicing*: the system takes an existing document or textbook and automatically divides it into slices. Each slice represents a piece of information about a topic – for example, a definition or a problem. The relationship between slices is partly inferred from the structure of the original document, for example the references to different sections. However it is usually updated manually, and other metadata, such as keywords, are also added manually to each slice. The slices are then reassembled in different levels of detail for individual students, depending on their level of knowledge and the scenario they wish to use the book for. For example, students can examine all the exercises for a topic to revise for an exam, or can find all the references to further literature.

2.3.5 ActiveMath

ActiveMath [MAB$^+$01] is a learning environment that holds a very detailed semantic model of the mathematics it teaches. This mathematical knowledge is kept in the OMDoc format

[Koh00]. However, as well as describing the mathematical relationships between concepts, the OMDoc documents in ActiveMath also contain *pedagogical* metadata [MBG$^+$03] – for example, the *abstractness*, *difficulty*, and *learning-context* of a concept.

Students are also modelled in detail. Like Intelligent Tutoring Systems, ActiveMath maintains an ongoing estimate of each student's understanding of the concepts that it teaches, but it also models students against eight *competencies*. For example, being able to understand a diagram is a different competency from being able to construct a mathematical argument.

ActiveMath uses these detailed models to generate courses that are tailored to individual learners' goals, competencies, and preferences. It can also develop courses for the same material using different pedagogical styles: for example, in a German teaching style the definitions and theorems might be presented before the examples, whereas for an American teaching style the examples might be presented first.

A comparison between ActiveMath and my research is provided in Section 9.7.

2.3.6 Adaptive hypermedia

Systems that alter online materials to make them more suitable for individual users are called *adaptive hypermedia*. This is a very active research area, as well as a technique that is often applied to Web-based Learning environments (for example the ActiveMath and Living Book systems described above). Brusilovsky, one of the developers of ELM-ART, provides an overview of how adaptive hypermedia and education have been linked [Bru00]. He also distinguishes *adaptive content* from *adaptive navigation support*. In adaptive content, the text of the content is altered for individual readers; in adaptive navigation support, the links are altered, redirected, or hidden to lead each user to the most appropriate content for them.

Adaptive hypermedia systems can also differ according to whether they alter the content at a page-level or a more fine-grained level, the particular techniques they use to adapt the content [Bru96], and the user model that they base their decisions upon [FMMCM04]. Some recent work has also examined *socially adaptive navigation* [BCF04] – personalising navigation support based on the navigation patterns of previous users. There are, however, too many adaptive hypermedia projects to describe them individually in this dissertation.

2.3.7 Summary

Many different systems have been developed to adapt course materials to individual students. Often these generate material from a single "authoritative" ontology or master text. However, the more complex or intricate the mechanism that alters the material, the harder it is for students to understand how or why the material has been altered. For example, Kay [Kay00] described how it is important for student models to be *scrutable* – that is, the students should be able to see and understand how the system is modelling them.

Where content is adapted in a fine-grained way (rather than at a page level), it raises the question of how students can refer to content when talking to each other – there needs to be some mechanism for the students to be able to look at the same version of the same content. Also, if the system relies on a detailed ontology or requires a large amount of metadata about each content item, it might be difficult for students (or even co-authors) to contribute to the book. Each author would need a detailed knowledge of the metadata scheme before he or she could contribute. Masthoff [Mas02] developed an "authoring coach" to teach authors how to provide the metadata.

While many Intelligent Tutoring Systems have been designed that allow students to work on diagrams or in graphical notations, the questions in most learning environments use much simpler interacton, often text-based. The focus on detailed student modelling also means that Web-based Learning Environments generally do not support Reactive Learning Environment questions – where the particular skills to model the student against are unclear.

2.4 Editing mathematics

Editing mathematical notation with a computer is a difficult interaction task. Written mathematics has a very large number of symbols, more than can be represented directly on the keyboard, and many of those symbols are usually arranged in a two-dimensional syntax. In this section I describe the various approaches that have been taken to support the editing of mathematics, particularly in educational settings.

2.4.1 Parsed text

One approach is to ask the user to write the mathematics using a different text-based one-dimensional syntax. This is particularly common in systems that were developed before graphical user interfaces became widely available. For example, the $\LaTeX$ typesetting system includes a text formula language and most UNIX installations include the `eqn` program that formats equations for the `troff` typesetter.

Raggett's and Batsalle's EZ-math system [RB97] uses a text language to allow maths to be written easily for use on the Web. The language is based on how mathematics is read aloud, because that is necessarily a one dimensional language using words. For example "`limit as x tends to a of function f(x)`" would produce "$\lim_{x\to a} f(x)$". EzMath elaborates the notation slightly, for example allowing brackets to be used to resolve ambiguities.

2.4.2 Mathematical sketching

Another approach is to allow users to handwrite mathematics using a stylus. MathPad^2 [JJLZ04] is an application for *mathematical sketching* (handwritten mathematics that can be associated with sketched diagrams). To simplify the parsing process, MathPad^2 requires the user to draw a lasso around expressions he wishes to be parsed, rather than automatically parsing the whole page. Parsed characters in the expression are rewritten using training examples of the user's own handwriting. This makes any mis-parsed characters obvious, but also preserves the look, feel, and spatial relationships of the handwritten mathematics. Diagrams can be sketched and linked to the expressions by labelling parts of the diagram with variables from the expression. MathPad^2 can then "rectify" the diagram as the variable's value changes – so for instance if an angle is labelled with a variable, then the angle in the sketch can be altered to match the variable value. If a drawing element is associated with a function of time, then the diagram can be animated. Supported animations include translational movement, rotation, and changing the value of an arc. Matlab$^{\text{TM}}$ is used as the computational engine for the system.

2.4.3 Structured Editing

Most WYSIWYG mathematics editors, however, use a technique called “syntax-directed” (or “structured”) editing. A set of menu options or buttons can place a template of a mathematical structure on the screen, which can then be filled in by the user. The most commonly used mathematical editor, the Design Sciences editor that is included in Microsoft Word, uses this technique. A screenshot of the BrEdiMa [MN06] system, which is a mathematical editor built in Javascript and HTML, is shown in Figure 2.6.

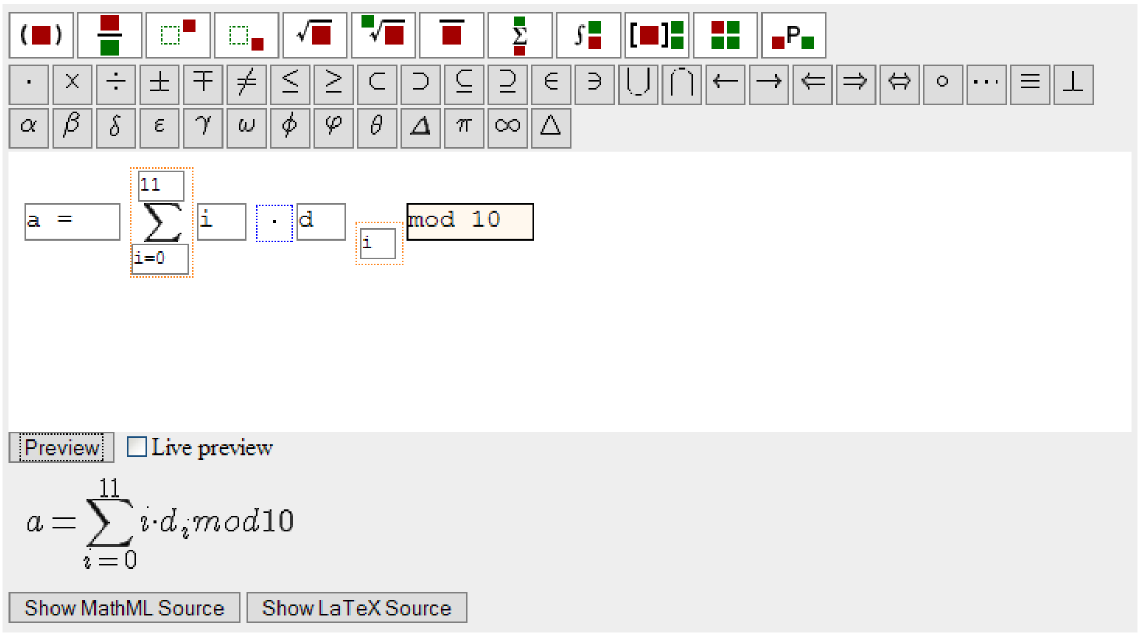


Figure 2.6: The BrEdiMa Web-based mathematical editor

Structured editing has a long history and was originally designed for writing computer programs. As early as the 1970s, systems such as EMILY [Han71] and the Cornell Program Synthesizer [TR81] allowed programs to be constructed by choosing syntactic templates in a top-down manner, rather than by typing text to be parsed. Recently, GNU TeXMacs [VDH01] has applied the technique for WYSIWYG editing of mathematical and TeX documents.

Structured editing has been found to help novices work with an unfamiliar programming syntax – the novice is guided by menus of legal operations, and syntax errors become impossible to make. The Carnegie Mellon programming environments [MPMV94] pioneered this use for the technique in the 1980s, and the Alice2 programming environment [KCC+02] is a more recent example. A number of toolkits for building structured editing environments have also been designed – for example, Harmonia [Bos01] and Barista [KM06].

2.4.4 Summary

Structured editing is the most common technique for editing mathematics because it supports the two-dimensional nature of maths and only requires a mouse and keyboard. Many students do not have styluses, so sketching can currently only be a niche solution. Text-based syntaxes have the limitation that students must learn the text syntax as well as the syntax of the mathematics it produces.

However, traditional structured editors are often too rigid to be ideal for education. The granularity of editing is usually fixed at a syntax level. For example, a teacher cannot group together semi-constructed pieces of mathematics that the student cannot break apart. In most systems the syntax of the mathematics itself is also fixed and cannot vary from document to document. Different questions in a textbook, however, might involve different structures, notations, and occasionally informal shorthand notations. Finally, it is often awkward to sketch out fragments of an answer, as each fragment must be created as a separate equation. So editing can involve cutting and pasting between multiple documents.

2.5 Educational Systems for Mathematical Proof

In this section I describe systems that have been specifically designed to support the teaching of proof and logic.

2.5.1 EPGY

The EPGY Theorem Proving Environment [SN04] is a stand-alone proof environment used in a number of courses at Stanford University. Students begin with a set of given statements and a proof goal. A menu based system allows the student to apply built-in strategies and inference rules to goals in order to build up a proof – this aspect of the system is intended to encourage “structured theorem proving”. Additionally, students can enter their own intermediate goals using a formula editor, and the proving environment will attempt to verify these goals using the Otter automated theorem prover.

2.5.2 DIALOG Project

The DIALOG Project [BHL+06, BHKK+07] is an ongoing project developing a system that can discuss proofs with students in natural language. The principles behind their philosophy are similar to those behind AutoTutor [CHG+00]. Human-to-human tutorials have frequently been found to be an effective teaching technique, so they wish to carry the pedagogy from those human tutorials across to automated tutorials. The proof domain the project has most examined is naïve set theory.

2.5.3 Diagrammatic Theorem Proving

Dr Doodle [WBG04, WBGJ02] is a diagrammatic theorem prover from Edinburgh University, specifically supporting *metric-space analysis*. It was developed out of the assumption that a significant number of students find reasoning diagrammatically easier than reasoning in formal mathematical notation. The diagrams show example objects and the relations between them. The *rewrite* rules that are the mainstay of theorem provers become *redraw* rules in Dr Doodle: rather than testing properties of the algebra and creating a new line of mathematics, they test properties of the drawing and create an appropriately altered drawing as the next step.

2.5.4 Systems for Propositional Logic

A number of educational systems have been designed for propositional (or sentential) logic. The Carnegie Mellon Proof Tutor (CPT) [SS94], the The P-Logic Tutor [LLB02], and Logic-

ITA [LY02, Yac03, Yac04] are all examples of Intelligent Tutoring Systems designed to teach propositional logic. CPT uses a combination of Fitch diagrams and a Goal Tree to describe the proof being developed. Logic-ITA represents proofs fairly simply – as a sequence of proof lines in a table – and focusses instead on detailed and effective modelling and assessment of the student. It assesses the validity of proof steps as the student works on them, and once the proof is complete returns to assess the usefulness of each of the steps. P-Logic Tutor doubles both as a tutor and as a research environment for tracking student learning and exploring the cognitive issues involved.

ETPS [ABP^+04] assists students in writing and checking formal proofs in propositional logic. The student asks ETPS to apply particular rules of inference, and ETPS handles writing the mathematics. Ehrensberger's and Zinn's DiaLog system [EZ97] treats propositional logic as a game between a proponent and an opponent. Proving a thesis is correct involves demonstrating that the proponent has a winning strategy that can successfully defend against any possible attack from an opponent. The user plays the part of the proponent, while DiaLog ensures that all possible alternatives of the opponent are considered. Hyperproof [BE94] teaches students the principles of analytical reasoning and propositional logic in the blocks world of Tarski's World.

Tutch [ACP01] is a tutorial proof checker that does away with proof environments completely and requires the proof to be written in a human-readable text-only syntax. In its goal to provide a human readable formal proof syntax, it is similar to the Isar language that the MathsTiles proofs in our system are translated to, but designed specifically for education.

2.5.5 Summary

There appear to be a wide variety of educational proof systems for domains where automated techniques can reasonably be expected to find an answer without human intervention, for example propositional logic. There are comparatively fewer systems for "harder" domains, such as Number Theory. The EPGY Theorem Proving Environment is the most relevant system in that regard. EPGY permits students to complete the proof by applying tactics from a menu rather than requiring them to write each line of proof themselves. As will be described in Chapter 6, this interaction style can lead to students gaming the system by trying each tactic in turn until some progress appears to be made. (I am not aware of any studies that have specifically investigated "gaming" behaviour within EPGY, however.)

2.6 Design Guidelines for an Intelligent Book

An Intelligent Book is a similar concept to a Web-based Learning Environment, in that it contains content integrated with appropriate exercises. (Arguably the definitive distinction is that an Intelligent Book restricts itself to the role of a textbook and does not, for example, grade students or check that required exercises have been completed.) However, in order to fulfil its role as a textbook, the Intelligent Book architecture developed in this dissertation is designed to meet certain goals:

1. **Graphical interaction with detailed advice**
   An Intelligent Book should be able to support the graphical interaction and detailed advice that can be found in many Intelligent Tutoring Systems and some automated question systems.

2. **Support for a variety of questions**
   As described in Section 1.2.1, an Intelligent Book should be able to have the wide variety of questions that one expects to find in a textbook. So, the architecture should support different graphical notations and different modelling or reasoning systems. It should not be restricted to a single pedagogical technique – as described in Section 2.2.4, different pedagogical techniques can be appropriate for different questions.

3. **Reactive Learning Environment questions**
   An Intelligent Book should be able to support Reactive Learning Environment exercises. These are suitable for design tasks and questions where the solution procedure is not known in detail, but have not generally been supported in Web-based Learning Environments in the past.

4. **Support for existing models**
   Many of the modelling systems used in tutoring systems, for example in Model Tracing tutors, are bespoke systems designed for education. However, an Intelligent Book should also be able to make use of existing modelling or reasoning systems rather than requiring every system to have been designed specifically for the book.

5. **Support for multiple explanations**
   In April 2007, the booksellers WHSmith listed thirteen different textbooks for thermodynamics as being in stock, and twelve more as available on order. Most university libraries do not limit themselves to a single text on a subject, and most courses' reading lists include more than one book. Many students do not limit themselves to textbooks but also use Wikis and Web-based tutorials. There is clearly not a single authoritative ontology or explanation for each topic, but a marketplace of competing explanations. An Intelligent Book, then, should not limit itself to a single explanation of a content item. Students and co-authors should be able to add alternative explanations and improve existing explanations during the life of the book.

6. **A content model that is extensible by students and automatically referable**
   If students are to be able to contribute to the book, then the content model should be reasonably straightforward. Students should not have to learn a detailed ontology or be taught how to write detailed metadata before they can contribute. The content model should, however, allow the automated advice from questions to refer to the content.

As well as designing a suitable architecture for Intelligent Books, this dissertation also seeks to develop proof exercises suitable for an introductory Number Theory course. Chapter 6 describes the design goals for these exercises.

CHAPTER 3

Supporting Complex Graphical Questions at the Client

This chapter introduces complex graphical questions and considers how to present them at the client. Section 3.1 gives an overview of how the client components are organised. Section 3.2 presents an electronics question as an example. Section 3.3 gives some more technical details on a simple method for building applets to support teaching through graphical notations. The work described in this chapter was carried out in 2003 and forms part of two papers that were published in 2004 [BRAH04, BB04].

3.1 Overview

When students are working on questions in an intelligent book, they should be able to use the appropriate notations for the subject matter. For example, a student working on an electronics question should be able to work with a circuit diagram. Digital electronics questions might involve timing diagrams or state charts; mathematics questions are likely to involve proofs written in mathematical notation. We also want the system to support progressive evaluation and be able to give students feedback while they are working, rather than always waiting for a "submit" button to be pressed.

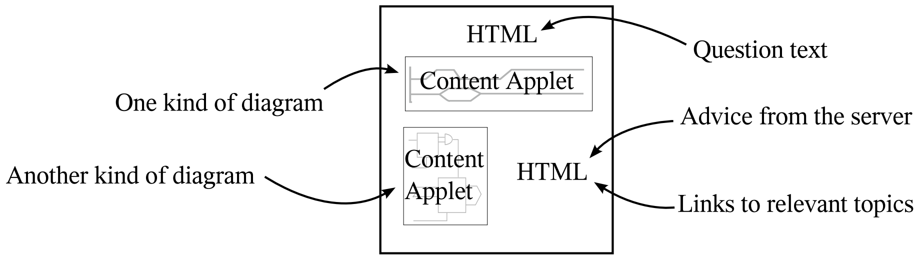


Figure 3.1: An exercise page may contain any number of graphical notations that the student works with – in this example, two diagrams. Comments from the system as the student is working may involve mark up on the Content Applets, alterations to the HTML on the page, or both.

Figure 3.1 shows a stylised diagram of an exercise page. The content applets represent the different graphical notations the student has to work with. For many kinds of notations, it remains costly and impractical to write an editor that only uses the HTML and Javascript that a browser can natively display, so the applets are implemented in Java. However, where the system makes text comments about the student's work or provides links to related content, we would prefer it to use the full HTML capabilities of the browser rather than a limited HTML component included in an applet. We therefore have a need to update the HTML of the page in-place, because performing a fresh page load would force the content applets to reinitialise. Since Google Maps was released in April 2005, there has been a lot of industrial interest in updating Web pages in-place using a JavaScript and XML technique that has since come to be known as AJAX [Gar05]. My architecture predates this popularisation of AJAX, and the coining of the term, but similarly uses a component to fetch data from the server and then alter the current page. Figure 3.2 shows the architecture.

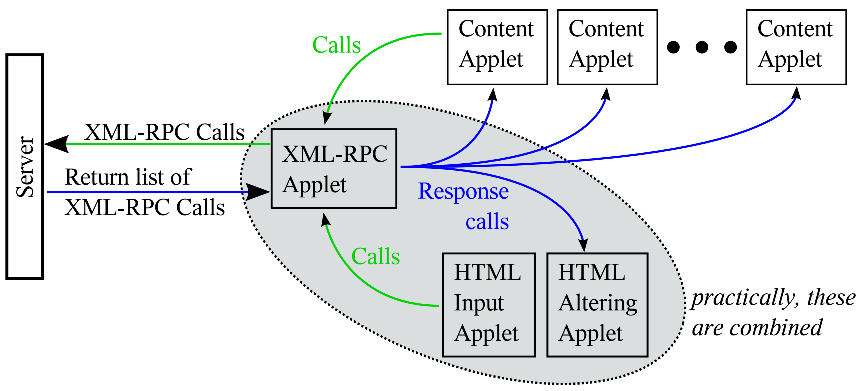


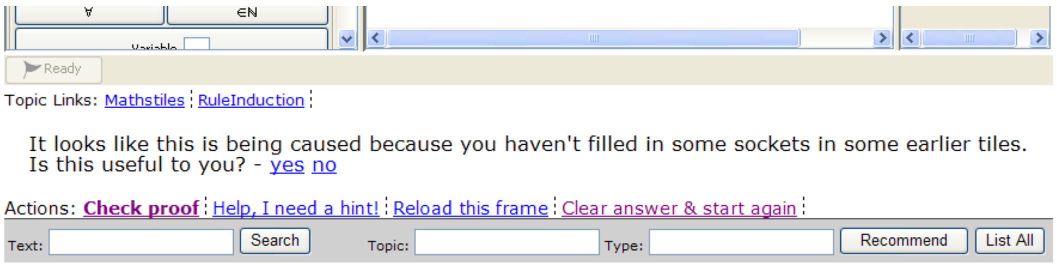
Figure 3.2: Calls are always initiated by the user, either through interacting with the Content Applets or the HTML Input Applet, which accepts Javascript calls from controls on the page. These are sent to the server as XML-RPC calls. The response is a list of XML-RPC calls the server wishes to make on the client in return.

A hidden Java applet handles communication with the server using XML-RPC [Win99]. For teaching applications, it can be helpful to script the client's behaviour from the server. This allows question authors to mark up the same content applet (the same graphical notation) in different ways for different questions, and lets them change many aspects of the system's teaching behaviour without altering the client components. However, to avoid the overhead of maintaining open connections between the client and the server and dealing with reconnects and timeouts, we would like all communication to happen in a call-and-response manner driven by requests from the client. To satisfy these two desires, the architecture requires the server's response to an XML-RPC call from the client to be a list of the XML-RPC calls it wishes to make on the client in return. These calls could be to the Content Applets to annotate or alter the student's work, or they could be to the HTML Altering applet that makes changes to the page's HTML on the server's behalf. The HTML Input applet accepts Javascript calls from controls on the page (which may have been placed by an earlier server response), and either applies them to the Content Applets or passes them on to the server, as appropriate. In this way, communication is always initiated by the client, but the client's teaching behaviour is completely scripted from the server.

Practically, the XML-RPC, Student Input, and HTML Altering applets need to be combined. The reason for this is that a call chain of $Javascript \longrightarrow Applet \longrightarrow Applet \longrightarrow Javascript$

can deadlock some browsers with some versions of the Java plugin. This would occur, for instance, if a Javascript link called the Student Input applet, which called a separate XML-RPC applet, which made a response call to a separate HTML Altering applet, which internally uses Javascript to alter the page.

Rather than arbitrarily altering any HTML on the page, the HTML Altering applet reserves a number of areas for particular kinds of interaction. A“system text” and “system HTML” area are kept for transient comments and controls from the server. An “actions” area is kept for permanent actions the student may wish to take, and a “topic links” area is kept for content links that relate to the exercise. These are shown in Figure 3.3. Organising the explanations area in this way keeps the interaction consistent, as the student is less likely to overlook a change to the page if they always occur in the same area, and also makes programming the interaction more consistent, as the dynamic area can be cleared after each student action to remove old comments.



Topic Links: Mathstiles ¦ RuleInduction ¦

It looks like this is being caused because you haven't filled in some sockets in some earlier tiles.
Is this useful to you? - yes no

Actions: **Check proof** ¦ Help, I need a hint! ¦ Reload this frame ¦ Clear answer & start again ¦

Text: Topic: Type:

Figure 3.3: The layout of the dynamic HTML area of the page. The topic links and actions for the question are grouped so that the student knows where to find them. The central area contains a “system Text” and a “system HTML” area. Functionally, there is no distinction between them, but separating them can make it slightly more convenient for calls from the server to set a prompt and then determine any appropriate HTML controls to show.

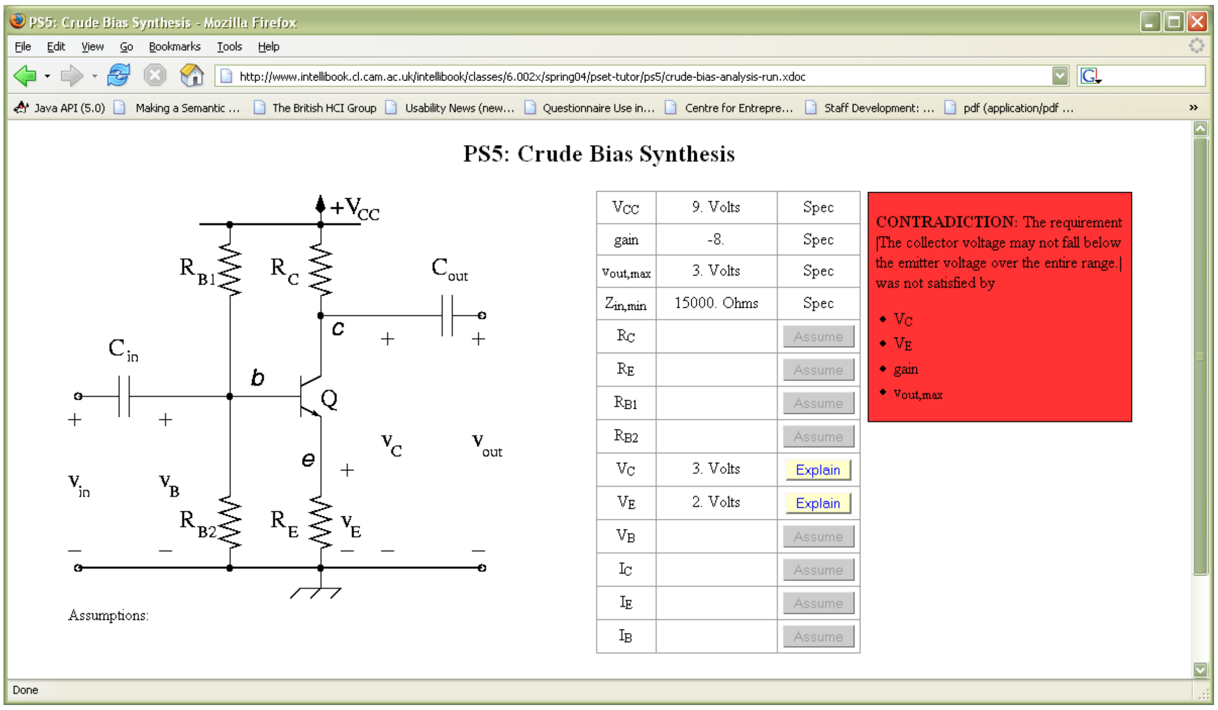
An interaction history log is also kept, although it is usually hidden. On an early version of the electronics question, this log could be exposed and showed the “command line” format of changes the student had made using the content applet. A command line entry box was also provided, with the intention that the interaction history log would teach the user how to use the command line box, in the style of Slator *et al.* [SAC86]. This style of interaction would still be useful for dialogue-oriented questions (the teaching methodologies of SOPHIE[BBB75] or AutoTutor[CHG$^+$00]), but the questions I describe in this dissertation prefer direct interaction with some work in a graphical notation over indirect conversations about that work.

Where possible, the system does not automatically make changes to the student’s work (for instance applying a correction). The technical reason for this is that if the student takes an action, there could be a network delay before the response comes back from the server. If this response changes the student’s work, then this new change could interfere with actions the student is currently taking, which could be frustrating. Instead, the system will often make its suggestion in text in the System Text area, and place a Javascript-backed link in the System HTML area. Clicking this link invokes the Javascript which will make the change on behalf of the student. This HTML area can also be used to offer alternative courses of action, or to ask mini-questions during the exercise.

The graphical nature of the exercises raises the issue of accessibility. The comments and feedback that are marked up in HTML on the page are automatically available to the browser’s own accessibility features. Making a Content Applet accessible, however, requires programming effort on the part of its author. The Java Runtime Environment provides an Accessibility API, which applet authors can use to make their Content Applets more accessible. It would also be possible to provide alternative Content Applets to some users. For example, a blind user might find a graphical circuit diagram to be awkward to use, and might prefer to use a Content Applet that presents the same circuit (using the same data model) in a different way.

3.2 A Question in Electronics

In 2003, I designed a client to support a type of electronics question that had been developed by Abelson, Sussman, and Hanson at MIT. This type of question had originally been asked using static HTML forms, but anecdotal evidence showed students were having difficulty following the explanations of errors given by the teaching system. The student is given the diagram of an electrical circuit and a set of requirements that it must meet. He or she must then set currents, voltages, and component values on the diagram in order to fully specify the circuit. All answers that obey the rules of electronics and meet the requirements are accepted as correct. A screenshot of a question using the original forms interface is shown in Figure 3.4.



PS5: Crude Bias Synthesis

V_{CC}	9. Volts	Spec
gain	-8.	Spec
$v_{out,max}$	3. Volts	Spec
$Z_{in,min}$	15000. Ohms	Spec
R_C		Assume
R_E		Assume
R_{B1}		Assume
R_{B2}		Assume
V_C	3. Volts	Explain
V_E	2. Volts	Explain
V_B		Assume
I_C		Assume
I_E		Assume
I_B		Assume

CONTRADICTION: The requirement |The collector voltage may not fall below the emitter voltage over the entire range.| was not satisfied by

- V_C
- V_E
- gain
- $v_{out,max}$

Figure 3.4: An electronics question as originally asked (using forms). The student is given an electrical circuit, in this case an amplifier, and is asked to choose component and property values in order to meet a set of specifications. Students had difficulty following the explanations from the server.

The reasoning system that supports this question is a *constraint propagation and truth maintenance system* [SS77], or TMS for short. The TMS makes deductions based on a *relation* (or *constraint*) model of the circuit. For example, in a circuit node where three wires meet, KirchhoffâĂŹs Current Law imposes the relation that the three currents entering the node must sum to zero. If two currents are set, the TMS will deduce the third; if all three are set, it will signal a contradiction if they do not sum to zero. Deduced values are propagated into other relations to make further deductions. The specifications from the question are set as constraints in the TMS, and whenever the student sets a value on the circuit, this value is set as a constraint in the TMS. So, a value set by the student may cause further values to be deduced, and it may also cause a contradiction to be flagged. The student is not forced to resolve the contradiction immediately – he can continue to set other values – but must do so eventually in order to complete the question. The question is complete when there are no unknown variables left in the circuit and no contradictions.

The TMS works from a hierarchical description of the circuit that is written in a Scheme-based language. Every circuit element ("*part*") has *terminals*, *parameters*, and *relations*. A terminal has a *current* and a *potential*. The relations may involve the terminal currents, the terminal potentials, and the parameters. For example, the resistor part-type has two terminals; the current into the two terminals are related by Kirchhoff's Current Law, and the difference in the terminal potentials is related to the currents and the resistance parameter by Ohm's law. Parts can also have different *models*, with different relations in each model. Transistor part-types tend to have a *bias model*, which describes their steady-state behaviour, and an *incremental model* which describes how they respond to transient signals. The circuit language is hierarchical. A transistor amplifier is composed of parts (a transistor, resistors, and capacitors), but it is also a part itself and can be used in larger circuits. As a part, it has its own parameters and relations; for instance its gain parameter relates the signal output to the signal input in the incremental model.

Anecdotal evidence suggested that students had difficulty understanding the contradictions that the TMS had flagged. These can involve several deduction steps, and the HTML forms client was only capable of displaying the final step. An example of the TMS's raw output is shown in Figure 3.5. When examined on the server, some of the TMS's explanations appeared quite difficult to follow as they can be overly detailed. For instance, if a line of resistors are in series and the current into the first resistor is set, the TMS will individually deduce the current into and out of each terminal of each resistor, where a human would simply mark a single current through the entire series.

To make the automated explanations more easily comprehensible, I designed the client to use a separate diagram model of the circuit. This diagram represented the "desired mental model" that the student should have of the circuit, and only showed the currents and potentials that the teacher wanted the student to talk about. The TMS's explanation trees were then automatically pruned and collapsed so that only variables marked on the diagram were included. (The pruning is carried out at the server, and is determined by whether there is a mapping from the server variable path to a client variable path, rather than strictly by examining the client circuit model).

A screenshot of the client is shown in Figure 3.6. The circuit diagram is a Java content applet, and the surrounding details are described in HTML. The actions of setting or clearing circuit values are taken directly on the circuit diagram, and any automatically deductable values are then marked in grey. Right-clicking on any of these grey values and then choosing the "how did you get this value?" option asks the server to explain the deduction. If a value the student has set causes a contradiction on the circuit, the value is marked in red, and text appears in the system text area to say that a contradiction has occurred. A link underneath the text then asks for an explanation.

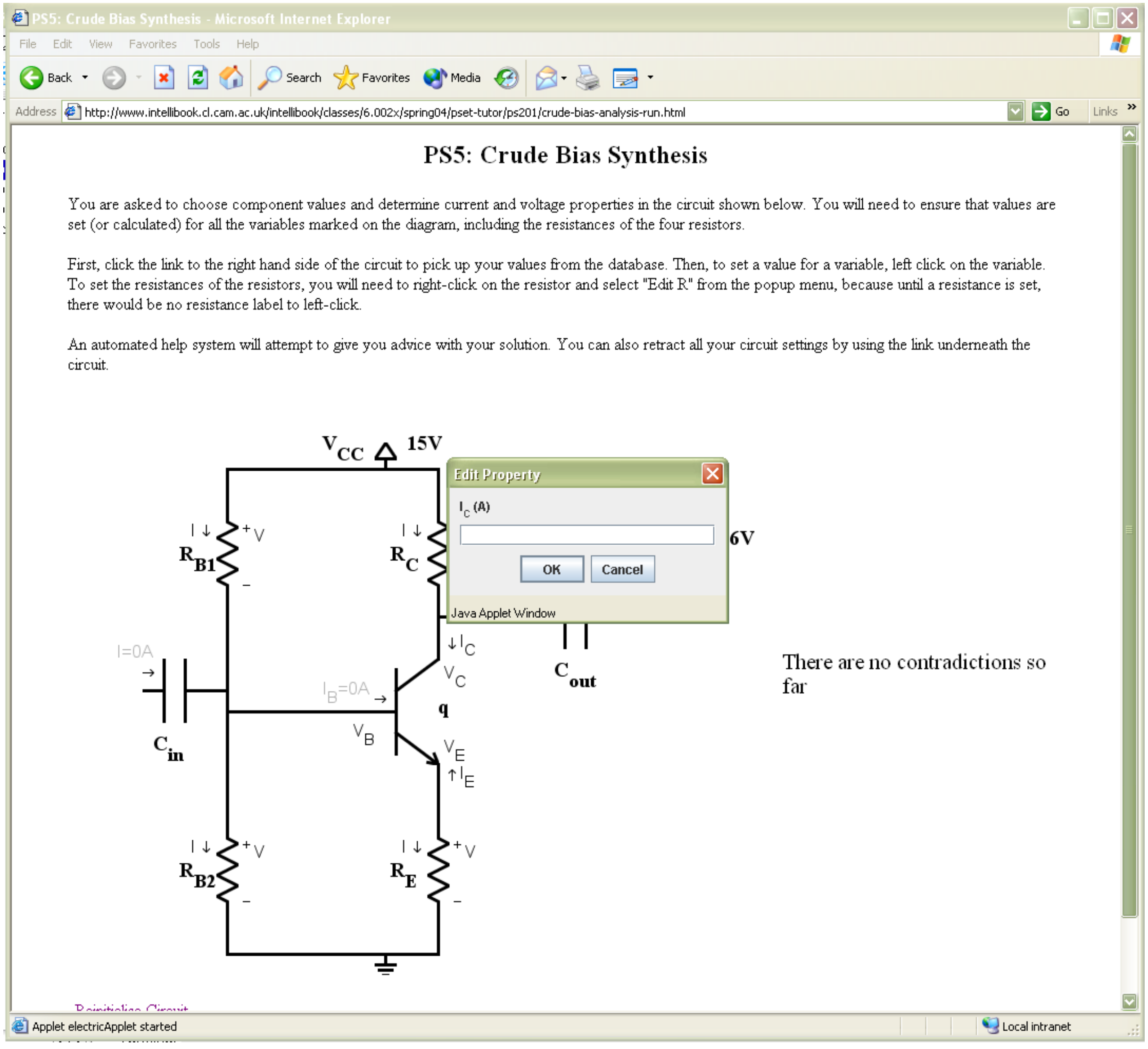
The explanations are shown on the diagram, starting with the final step. At each step,

```
(CEP118 contradiction found by (<swing-high) (CEP116 CEP75))
(CEP116 (v:lhs:swing-high) = 16 set by (+lhs:swing-high) (CEP67
  CEP114))
(CEP75 (potential vcc bias) = 15 set by (-rhs:kvl power bias) (CEP74
  CEP49))
(CEP67 (swing) = 6 set by assumption (CEP68))
(CEP114 (v:1:lhs:swing-high) = 10 set by (-1:lhs:swing-high) (CEP112
  CEP49))
(CEP74 (voltage power bias) = 15 set by (=rhs:voltage-source power
  bias) (CEP63))
(CEP49 (potential gnd bias) = 0 set by (v:rhs:bias-ground) ())
(CEP112 (potential c bias) = 10 set by assumption (CEP113))
(CEP63 (strength power) = 15 set by assumption (CEP64))
(QED)
```

Figure 3.5: The TMS explaining one of its contradictions. The step labels are highlighted in green; part of the deduction that can be considered to be the rule is highlighted in red. In this case, the TMS is complaining that if the output bias potential is 10V and the output swing is specified as 6V, then the voltage needs to swing above the voltage of the power rail (15V).

the values involved in this step are highlighted in red, and text describing the rule and the other variables that caused this deduction is displayed. Variables that are not involved in this deduction step but are involved in other steps are highlighted in orange. Links for each of the variables involved in this deduction step allow the student to see how those variables got their values in turn. These links either lead to other deduction steps, or simply tell the student that the value was set directly by the student or in the question specifications. By clicking through the links, the student can navigate the (pruned) TMS deduction tree. Additional links underneath navigate through the tree in a flattened ordered manner, effectively allowing the student to animate the deductions on the diagram. A screenshot of a deduction step is shown in Figure 3.7.

Finally, it is worth discussing the lack of a student model. Many tutoring systems rely on a detailed model of the students' understanding of the rules of the domain, perhaps represented in a Bayesian network. So for instance if the student caused a contradiction that broke Ohm's law, the system would adjust its model to suggest that perhaps the student does not understand Ohm's law. In this question, however, the student never actually has the opportunity to apply rules such as Ohm's law – if a value can be deduced from simple rules, then the TMS sets it automatically. Also, contradictions often involve a chain of six or more deductive steps. It seems unreasonable to mark the student down on each of the six rules involved, just as it would be unreasonable to mark a student down on "understanding multiplication" because she can't calculate $593,421 \times 647,823$ quickly in her head. It is not the basic rules that are being exercised in the question, but experience working on a realistic problem with help from automated verification tools. Many of the questions addressed in this dissertation share these properties – there is not a transferrable straightforward process to model the student against, and it is not the basic "rules of the domain" that are being tested. Although we want the architecture of an Intelligent Book to support the


PS5: Crude Bias Synthesis - Microsoft Internet Explorer

http://www.intellibook.cl.cam.ac.uk/intellibook/classes/6.002x/spring04/pset-tutor/ps201/crude-bias-analysis-run.html

PS5: Crude Bias Synthesis

You are asked to choose component values and determine current and voltage properties in the circuit shown below. You will need to ensure that values are set (or calculated) for all the variables marked on the diagram, including the resistances of the four resistors.

First, click the link to the right hand side of the circuit to pick up your values from the database. Then, to set a value for a variable, left click on the variable. To set the resistances of the resistors, you will need to right-click on the resistor and select "Edit R" from the popup menu, because until a resistance is set, there would be no resistance label to left-click.

An automated help system will attempt to give you advice with your solution. You can also retract all your circuit settings by using the link underneath the circuit.

Edit Property

I_C (A)

There are no contradictions so far

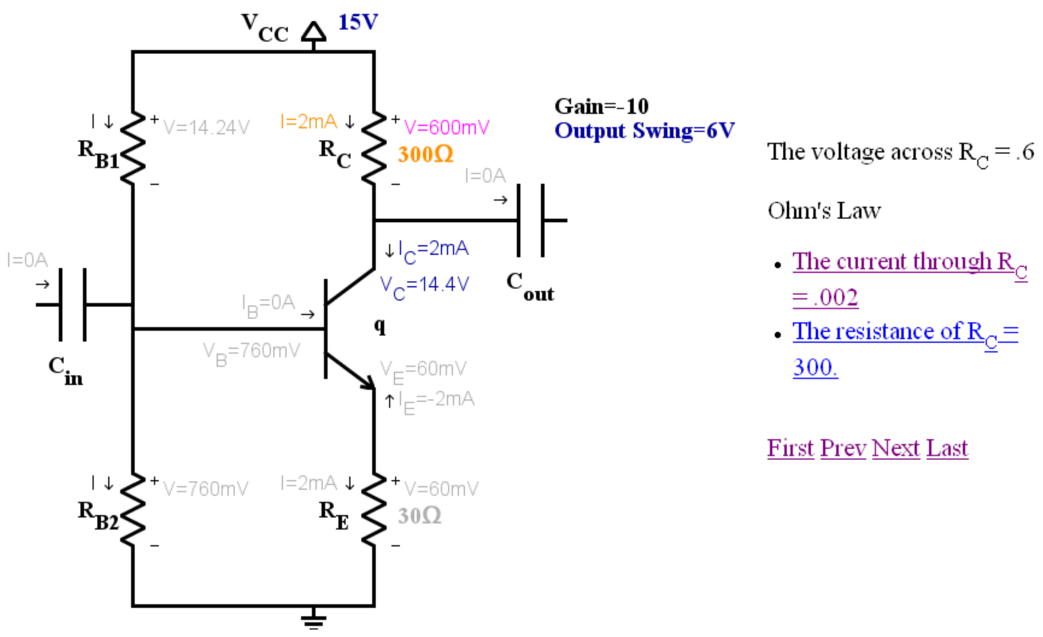
Applet electricApplet started

Figure 3.6: The electronics question using the updated question architecture.

more straightforward questions, the questions developed in this dissertation are mostly ones where the student has not been taught an exact process to follow to answer the question, and the system might not know one either.

3.3 Technical Detail

For an Intelligent Book to support a variety of different kinds of questions for a subject, it needs to be relatively quick and easy to write content applets for different kinds of graphical notations. In this section, I describe how the architecture supports fast construction of new content applets, through the `cam.cl.intelligentBook.domEditors` package.



The voltage across R_C = .6

Ohm's Law

- The current through R_C = .002
- The resistance of R_C = 300.

First Prev Next Last

Figure 3.7: A step of the TMS's deductions being displayed on the client. The variable being explained, the voltage across R_c, is highlighted in magenta. The other variables involved in the step, R_c and I_c, are highlighted in orange. Variables that are involved in other steps of the explanation are coloured dark blue. The variable links in the text allow the student to ask how those variables obtained their values. The links *First*, *Prev*, *Next*, and *Last* allow the student to step through the deductions in the explanation in order.

3.3.1 Cooperative XML Documents

Because we want the server to be able to comment on the student's work progressively, we consider the student to be working on a remote document rather than preparing an answer to be submitted. These documents are stored on the server as XML. The way the student interacts with the graphical notation, however, is essentially defined by the content applet on the client. It would be possible to write content applets as thin clients with only the server making changes to a document, but this would require two components (client and server) to be written to describe the GUI behaviour for each notation, and network latency could impair the quality of interaction. Instead, we treat the system as having two documents – a client document and a server document – that need to be kept synchronised.

Because we may wish to make changes to the student's work from the server or through Javascript links, it is important that the internal data structures in the content applet can support an API to update the document at runtime. Ideally, for maximum code reuse, this would be an API based on the standard Document Object Model (DOM) for XML. However, each content applet, being a different notation, will need its own application-specific classes to display the data in the document. XML serialisation technologies, such as JAXB[VF04], support generating application-specific objects from XML documents at initialisation time, but do not support DOM-based alterations to them afterwards. Programming support for the DOM interfaces into the application specific classes would require a large amount of work. So instead, we treat the XML as object field data that has been separated from its methods and behaviour. At load time, the XML is parsed into a standard DOM tree with no specific behaviour. This tree is then passed to a *behaviour factory*, which creates a *behaviour object* for each element, and attaches it to the element's `userData` field that is present in DOM Level 3, using the string "`behaviour`" as the data key. This is illustrated in Figure 3.8.

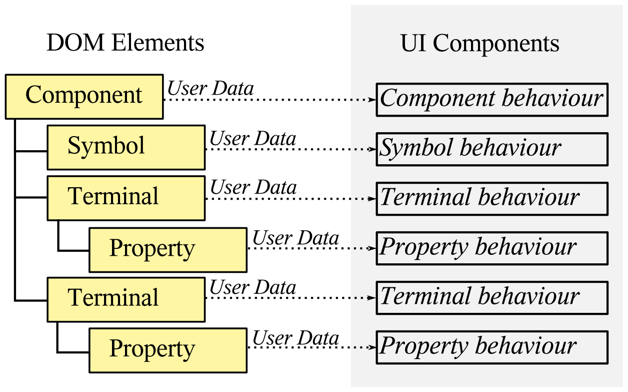


Figure 3.8: The UI components are attached as "behaviour objects" to the DOM elements, using the `userData` field. This allows the DOM to be used as the model in a poor man's *Model-View-Controller*. While DOM Level 3 does not support event notifications for simple updates, the external update API is made to call update functions on the behaviour objects by convention.

Essentially, this is a form of Model-View-Controller [GHJV95]: the standard DOM classes provide the model, and the behaviour objects provide the view and controller. Although the DOM Level 3 classes do not support an event listener to notify the view of updates, the external API is made to call update methods on the behaviour objects of each element that is altered.

Behaviour objects are required to implement the `ElementBehaviour` interface, which defines the `synchroniseFromElement` method for use by the external API. An abstract factory class, `ElementBehaviourFactory`, builds behaviour objects for an element and all its children. To define the mapping between elements and behaviour objects, content applets should subclass this factory and implement the `buildThisElementBehaviour(Element)` method.

3.3.2 Simple Change Format for XML

The external API the system uses to describe changes to the XML is a simple set of functions that I have dubbed *Simple Change Format for XML* (SCFX). The need to define a new API came from the unsuitability of the industry standard alternatives. XUpdate [LM00], which is used by a number of XML databases, is an XML dialect for describing changes to an XML document. However, it has not been consistently maintained for some years. Also, as an XML dialect, it needs to be processed into calls, and the size of the processor could be difficult in a content applet. An API, however, needs no processor and gets two usable written syntaxes for free: the XML-RPC representation and the Javascript representation of calls to the API. XQuery, which is a W3C[1] Recommendation, has recently gained update functions in its most recent draft [CFR06]. However, as a scripting language it would also need a sizable parser and processor (and in any case its update functions were added too recently for our development).

The methods in SCFX are shown in Figure 3.9. A version of the SCFX interface that includes an extra `prefix` parameter, PSCFX, is also provided. The `prefix` can be used to give context to the operation. So, if a content applet holds more than one document, the `prefix` can

[1]World Wide Web Consortium

be used to choose between them. Other times it may be useful for the `prefix` to hold a root XPath within the document from which the other XPaths are evaluated.

The `with` method is provided for two purposes (although in practice our content applets have not needed it yet). The first is to avoid the need to repeatedly resolve the same XPath in a set of calls. The second is if there is a need to wrap a set of changes into an atomic transaction. The format for the calls to be made within the `with` method depends on the implementing application. For content applets, it is most convenient if these calls are formatted as Javascript, since the browser's Javascript parser is easily accessible. Where SCFX is used on the server, it may be appropriate for them to be in the same format as the original call – for instance if this call was made using XML-RPC, then the wrapped calls should also be formatted using XML-RPC. Again the reason for this choice is that a parser for that format must already be present.

I provide support for the SCFX methods in the `ScfxHandler` class. Content applets may or may not expose all of the methods – since content applets are usually written hand in hand with teaching scripts, it will depend on which methods the teaching scripts need. Naturally, content applets can also include their own specific methods. Because the XML-RPC applet works using reflection (runtime discovery of the available methods), these extra methods automatically become available for the server to call.

`setValue(`*`XPath`*`, `*`value`*`)`

`setAttribute(`*`XPath`*`, `*`attribute name`*`, `*`value`*`)`
Required because `setValue` cannot set an attribute that does not exist: the XPath would resolve to nothing.

`setAttributes(`*`XPath`*`, `*`list of attribute names`*`, `*`list of values`*`)`

`addFragment(`*`XPath`*`, `*`XML to add`*`, `*`child index`*`)`

`removeSubTree(`*`XPath to remove`*`)`

`removeSingle(`*`XPath to remove`*`)`
Children of the removed nodes are attached to the removed nodes' parents.

`moveSubTree(`*`XPath to move`*`, `*`XPath of new location`*`, `*`child index`*`)`

`moveSingle(`*`XPath to move`*`, `*`XPath of new location`*`, `*`child index`*`)`
Children of the removed nodes are attached to the moved nodes' former parents.

`replaceSubTree(`*`XPath to replace`*`, `*`new XML`*`)`

`replaceSingle(`*`XPath to replace`*`, `*`new XML`*`)`
Children of the removed nodes become children of the new XML if possible.

`with(`*`XPath`*`, `*`Further calls`*`)`
Performs the further calls on the nodes resolved by the XPath. This method could also be used to wrap transactions.

Figure 3.9: Simple Change Format for XML (SCFX). The format for the calls to be made within the `with` method depends on the implementing application. A prefixed version of SCFX (PSCFX) provides the same methods with an additional *`prefix`* parameter to provide context for the operation – for instance selecting which document to act upon, or providing a base XPath to operate from.

3.3.3 Document Management

Often, documents contain definition elements that describe how other elements should appear – for instance the electronics applet allows components to be defined and then instances of those components to be displayed. It can also be useful to break these definitions out into separate utility documents. A `DocumentSystem` class is provided to support this. It maintains the set of currently open documents, which are accessible by either the `name` attribute on the document's root element or the URL from which the document was retrieved.

When a document is loaded, the `DocumentSystem` looks for `requires` elements in the `DOMEditors` namespace, to see if any other documents need to be loaded. For example in the transistor amplifier question the following is used to load the document containing the symbol definitions for transistors, resistors, and other needed components:

```
<requires name="analog" uri="electronics/analogComponents.xml"
 xmlns="http://www.cl.cam.ac.uk/users/whb21/DOMEditors" />
```

Documents' DOM trees can be accessed directly, but two convenience methods are also provided:

`getBuiltElement(`*`element tag`*`, `*`name attribute`*`, `*`default document`*`)`
`getUnbuiltElement(`*`element tag`*`, `*`name attribute`*`, `*`default document`*`)`

The methods differ as to whether the `DocumentSystem` should ensure the behaviour objects for the element have been constructed before the element is returned. (Many elements, such as those describing default font size settings or metadata about the document itself, do not require behaviour objects.) Elements can be fetched from other documents by formatting the name attribute as *`documentName`*`:`*`elementName`*. A `DocumentSystem` also has an associated `XPathHandler`, which provides convenience methods for running XPath expressions on the document.

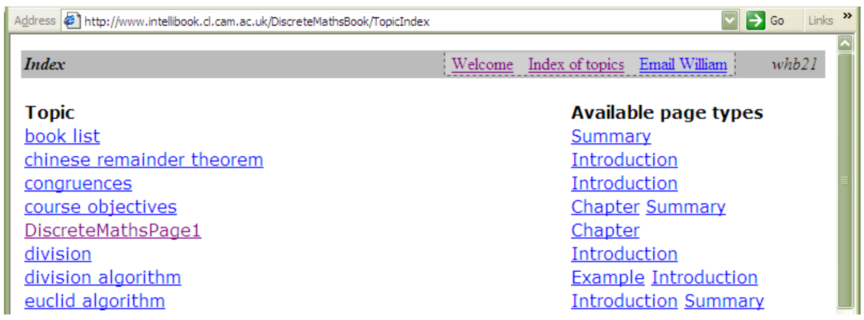
CHAPTER 4

Content Model

In this chapter, I describe the way that content is organised within the Intelligent Book. A number of previous online textbook systems have taken a strict semantic and ontological approach to content, such as OMDoc [Koh00]. However, a strict ontology could pose a barrier to readers wishing to add their own content – they would need to know the ontology in detail in order to fit their entry within it. The approach I have taken is to use an informal ontology that lets readers add alternative entries for topics, or even alternative chapters, more easily. Automated analysis of the book's content could then infer a more precise ontology if it was required. The content model is very simple both so that its complexity should not be a barrier to the readers' understanding of the book, and also to create the minimum necessary content model to support the exercises that are the main focus of this dissertation.

Section 4.1 gives an overview of how pages are categorised within the book. Section 4.2 describes the architecture and data model that support this. Section 4.3 describes how the book supports hierarchies such as chapters, sections, and subsections.

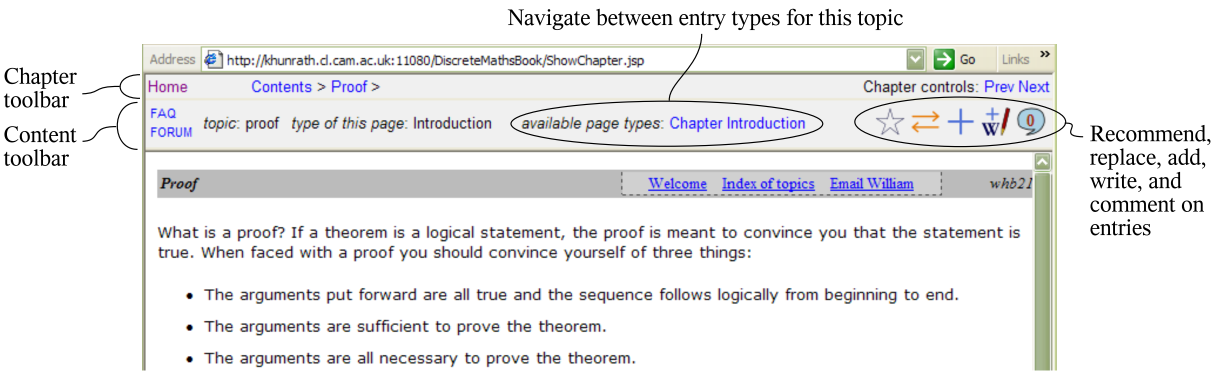
4.1 Overview of the Topic Structure


Address
http://www.intellibook.cl.cam.ac.uk/DiscreteMathsBook/TopicIndex
Go
Links
Index
Welcome
Index of topics
Email William
whb21

Topic	Available page types
book list	Summary
chinese remainder theorem	Introduction
congruences	Introduction
course objectives	Chapter Summary
DiscreteMathsPage1	Chapter
division	Introduction
division algorithm	Example Introduction
euclid algorithm	Introduction Summary

Figure 4.1: Pages in an Intelligent Book are classified by topic and type. This index page lists the topics and the available page types for each of those topics; there may be more than one entry per topic-type combination, in which case selection scripts choose which one to show.

Content in an Intelligent Book is classified by topic and by the type of entry. For example, a student could ask for an *introduction* to *mathematical induction* and then ask for an *example*. Figure 4.1 shows part of the alphabetical topic index of a book. There may be more than one induction example in the book, in which case server scripts choose an appropriate one to show. These scripts are configurable, and Section 4.4 discusses selection techniques they could use.


Navigate between entry types for this topic
Chapter toolbar
Content toolbar
Recommend, replace, add, write, and comment on entries
Address http://khunrath.cl.cam.ac.uk:11080/DiscreteMathsBook/ShowChapter.jsp
Go
Links
Home
Contents > Proof >
Chapter controls: Prev Next
FAQ
FORUM
topic: proof type of this page: Introduction
available page types: Chapter Introduction
Proof
Welcome
Index of topics
Email William
whb21

What is a proof? If a theorem is a logical statement, the proof is meant to convince you that the statement is true. When faced with a proof you should convince yourself of three things:

- The arguments put forward are all true and the sequence follows logically from beginning to end.
- The arguments are sufficient to prove the theorem.
- The arguments are all necessary to prove the theorem.

Figure 4.2: A content toolbar allows students to navigate between different types of entry for a topic, to recommend an entry, choose an alternate entry, add or write their own entry, or comment on the existing entries. A chapter toolbar allows students to navigate the topics of a chapter in an ordered manner.

Figure 4.2 shows a page of content within the book, and focusses on the toolbars that appear above the page. The lower of the two toolbars allows students to work with the book's content model in a number of ways; they can:

- navigate between different types of entry for this topic.

☆ recommend this entry. Or, if they have already recommended this entry, they can retract their recommendation.

⇄ ask for a list of alternative entries for this topic and type to choose from. This is presented as a simple list, with the title and summary of each entry along with informative metadata such as the author and who added the entry into the book.

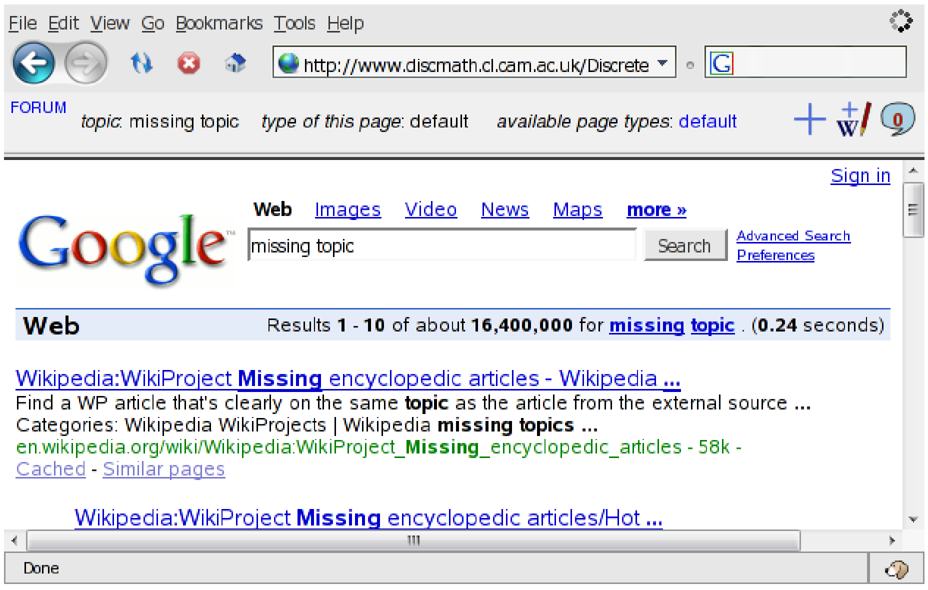
+ add an entry they have found on the Web for this topic.

W write their own Wiki-style entry for this topic. Student Wiki entries can refer to other topics in the book, and there can be more than one student Wiki entry for a topic.

💬 comment on this entry or read other people's comments.

FORUM link to a forum where they can discuss this topic with other students.

If the Book does not have an entry for a topic, then by default it performs a search using a popular Web search engine. In that case, the content toolbar will not show all of the options listed above – it is not possible to recommend an entry or ask for alternative entries if no entries exist. (It is possible to add the Web search as a *search* entry itself, but this does not happen automatically. The rationale for this is to make it more visible that there were no entries, and to encourage the reader to write or add their own.) The comment icon is present, in case readers wish to discuss what entries to add before they actually add an entry. The case where there are no entries returned is shown in Figure 4.3.


File Edit View Go Bookmarks Tools Help
http://www.discmath.cl.cam.ac.uk/Discrete
FORUM
topic: missing topic
type of this page: default
available page types: default
Sign in
Web Images Video News Maps more »
missing topic
Search
Advanced Search
Preferences
Done

Web Results 1 - 10 of about 16,400,000 for missing topic . (0.24 seconds)

Wikipedia:WikiProject Missing encyclopedic articles - Wikipedia ...
Find a WP article that's clearly on the same topic as the article from the external source ...
Categories: Wikipedia WikiProjects | Wikipedia missing topics ...
en.wikipedia.org/wiki/Wikipedia:WikiProject_Missing_encyclopedic_articles - 58k -
Cached - Similar pages

Wikipedia:WikiProject Missing encyclopedic articles/Hot ...

Figure 4.3: If no entry exists for a topic, a Web search is returned and the recommend and replace icons are hidden.

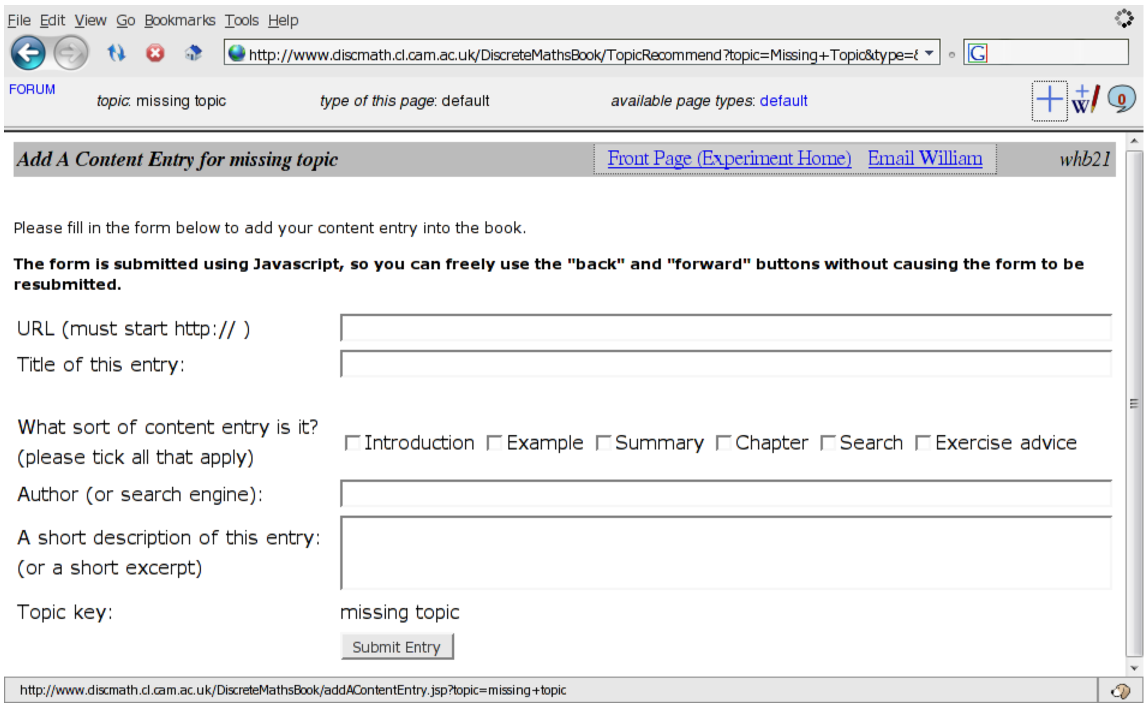
Adding a new page for a topic involves filling in a short form for the new entry. This is shown in Figure 4.4. The form for adding a new Wiki entry is similar except that it asks for the content of the page rather than the URL. It is worth noting that the reader cannot create new entry types, but has to choose from a predefined list. This is not a technical limitation but a practical one – if readers could create new types of entry at whim, then the list of entry types available for a given topic could quickly become so large that it would be unnavigable by other readers who look up that topic.

The default list of types are:

Summary	Brief, and assumes that the reader has some familiarity with the topic.
Introduction	Longer, and assumes that the reader has not encountered the topic before.
Example	A page describing a worked example.
Exercise	A live exercise within the Book that the student can try.
Exercise advice	A piece of advice about an issue that might come up in an exercise.
Search	Executes a search using a Web search engine.
Chapter	Takes the reader through the topic in an ordered manner. Chapters are made up of subtopics and are described in Section 4.3.

A content item can be listed under more than one type. The readers of the book are not able to add new exercises within the book because this would involve setting up the appropriate teaching script, content applet, and modelling or reasoning system, which it is not yet practical for a student reader to do.

Students can add new topics as well as adding new pages. There are two ways of doing


File Edit View Go Bookmarks Tools Help
http://www.discmath.cl.cam.ac.uk/DiscreteMathsBook/TopicRecommend?topic=Missing+Topic&type=
FORUM
Front Page (Experiment Home)
Email William
whb21
Submit Entry
http://www.discmath.cl.cam.ac.uk/DiscreteMathsBook/addAContentEntry.jsp?topic=missing+topic

topic: missing topic *type of this page*: default *available page types*: default

Add A Content Entry for missing topic

Please fill in the form below to add your content entry into the book.

The form is submitted using Javascript, so you can freely use the "back" and "forward" buttons without causing the form to be resubmitted.

URL (must start http://)	
Title of this entry:	
What sort of content entry is it? (please tick all that apply)	☐ Introduction ☐ Example ☐ Summary ☐ Chapter ☐ Search ☐ Exercise advice
Author (or search engine):	
A short description of this entry: (or a short excerpt)	
Topic key:	missing topic

Figure 4.4: Adding the URL of a new entry for a topic involves filling in a short form. Adding a new Wiki entry is similar except that the Wiki text rather than the URL is required. Readers cannot add an *exercise* for the practical reason that they cannot yet configure the teaching script, content applet, and modelling or reasoning system for the question.

this. The first is to invoke the Add Topic Entry form without a pre-specified topic; this is the way that the primary author would normally add content into the book. The second is to write a Wiki page (or other content page) referring to the new topic. When a reader follows the link, they will effectively perform a look up for a topic with no entries, and will have the option of adding or writing an entry for the topic. It is perhaps preferable for readers to use this method to add entries because it ensures that the new topic does not immediately become a disconnected island in the topic graph. Consequently, no links to the Add Topic Entry form are given which do not specify a topic (though the reader could easily work out the URL).

Accessibility issues in the content model are handled in a straightforward way. Both the chapter and the content toolbars contain plain HTML, and text equivalents are provided for all non-text elements (for example the image icons all have alternative text specified). These should therefore be available to the browser's own accessibility features. The toolbars are currently implemented using frames, which some assistive technologies handle better than others, but this could be reimplemented to compose the toolbars and content into a single HTML page without frames.

4.2 Architecture and Data Model

At the simplest level, the architecture of the content system breaks down into the four parts shown in Figure 4.5. A model database contains details of entries for each topic and type, and also a record of how readers have interacted with those entries. Selection scripts use this database to recommend particular entries to particular readers, and also update the database as readers interact with entries and add new pages. Some of the content entries within the book are editable using a slightly modified Wiki, while others are fixed or external resources. A description of how the selection scripts work with the database model is given in Section 4.2.1, and a description of the modified Wiki is given in Section 4.2.2.

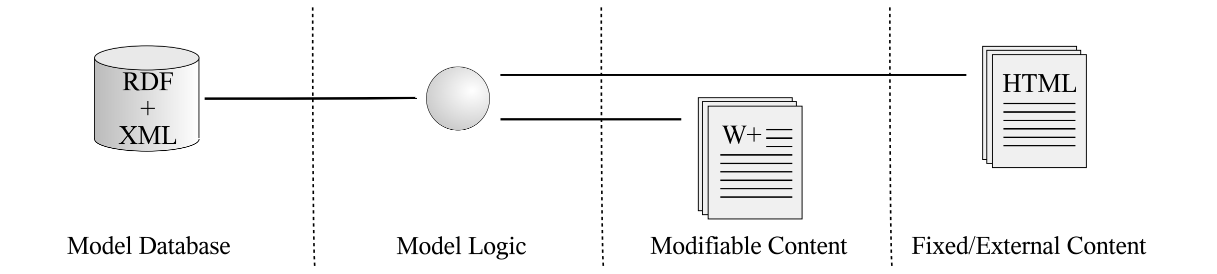


Figure 4.5: A simple overview of the content system

4.2.1 Database and Selection Logic

The XML database holds three distinct collections that relate to the content. The first collection holds RDF [BM04] data describing the entries that are in the book. Each entry lists the URL, the topic and type that the entry was added for, together with informative data such as the original author of the page. Entries are also separated according to which user added them to the book, for datamining purposes. Although the students can add content to the book that will be made available for other students, the primary author of the book can prevent students from crowding out the original entries in a couple of ways. Adding a $showFirst$ tag to a page's RDF entry ensures that the tagged entry will be shown first to a student looking up this topic for the first time, regardless of whether other entries have more recommendations for them. Adding a $noAdditions$ tag prevents the students from adding entries to that topic or creating Wiki entries for it at all.

The second section is a "student content model" which records how each student has interacted with the entries. When a page is shown, it is marked as $inbook$ for that topic for that reader. If the reader looks up the same again later, the $inbook$ page will always be shown. This prevents the book from behaving like a shifting sand, constantly changing while the reader is away from the book. If the reader rejects this page and chooses a different one, the old page is marked $rejected$ and the new page becomes $inbook$. The reader can also toggle on or off a $recommended$ tag using the recommendation button on the content toolbar. The recommendation model is kept very simple on the philosophical grounds that while a satisfied reader might or might not say that they are happy with a page, a dissatisfied or confused reader probably will ask for a different page instead.

The third section that is stored in the database is the comments board. Comments can be made both on particular pages and also on entire topic. It is kept in the same database rather than in separate commodity forum software with a view to future work. Potentially, automated analysis tools could examine the comments for pages and topics in order to mark those pages as suitable for particular kinds of reader.

I found it helpful to keep all three collections in the same generic XML database (rather than using a dedicated RDF database for the RDF data) partly for the simplicity of having to maintain fewer pieces of software, and partly because this allows XQuery selection scripts to query across all three sections very easily.

4.2.2 Modified Wiki

Readers can write their own pages for particular topics in a slightly modified Wiki system. Because the book supports more than one entry for a given topic, the Wiki should also support multiple entries for that topic. However, the Wiki should also be changed so that when a reader clicks on a WikiWord within a page it is treated as a topic look up within the intelligent book, rather than a direct link to another Wiki page.

In practice, this is very easy to implement with a small change to existing open source Wiki software. (In the Discrete Mathematics book that I constructed, I used JSPWiki.) The engine that generates HTML from the text source was altered so that WikiWords within the page would generate URLs that query the book's page recommendation system, but the Edit links and other administrative controls for this page would remain operating on this page within the Wiki. Since readers would no longer use WikiWords to access Wiki pages directly, the pages could then be stored in the Wiki using a simple unique ID. This is summarised in Figure 4.6

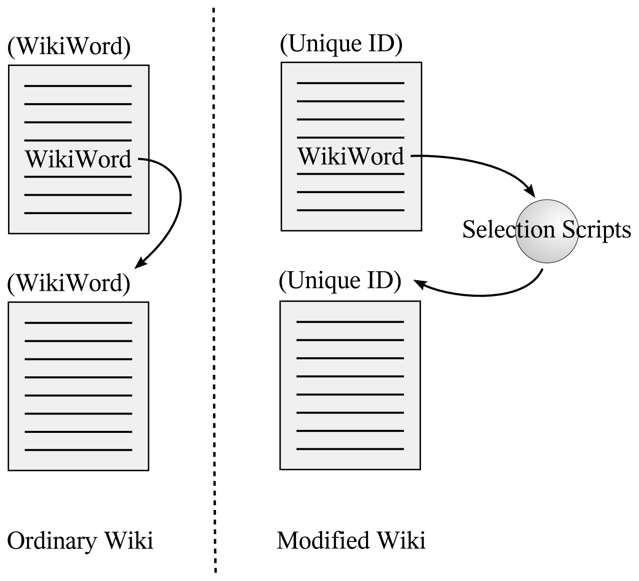


Figure 4.6: The Wiki is altered so that WikiWords link into the page recommendation system. The Wiki pages can then be stored within the Wiki under a unique ID rather than the WikiWord.

The intelligent book also provides its own search features, which index the Wiki pages, so the Wiki software's search features could be switched off.

4.3 Supporting Chapters, Sections, and Subsections

A textbook is not just a directory of entries for particular topics. It also contains chapters, sections, and subsections. The intelligent book supports this by allowing chapters to be defined as an ordered list of topics. The type of page to show for each topic can also be constrained. $Chapter$ is itself a page type, so chapters can contain other chapters, allowing an unlimited hierarchy.

Each chapter has a contents page, which is the first page of the chapter the student will visit and has the structure of the chapter embedded within it. Usually, the first page of an intelligent book is itself a chapter contents page, with each of the topics in its structure limited to the *chapter* page type. This gives an order to the top-level of chapters within the book, and it is this that allows readers to "turn to page one and start reading".

Embedding the chapter as a topic structure within its contents page has two advantages. Firstly it means that chapters can easily be stored in the RDF database simply as another entry for that topic. Secondly, it means that alternate chapters can be added to the book by writing a single page that could even be held in the modified Wiki. So, readers could write their own chapters.

In terms of implementation, the chapter structure is held in a hidden HTML form and the contents page includes a reference to a Javascript library. Most of the hidden form can simply be cut and paste into the page. The topic structure is written in Javascript Object Notation. An example based on the lecture notes for the Cambridge discrete mathematics course is shown in Figure 4.7.

```
< input type="hidden" name="chapterOutlineSrc"
  id = "ibChap_outlineInput"
  value=' { chapterName: "Integers", topics: [
    { topic: "sets", type: "Introduction" },
    { topic: "mathematical induction", type: "Introduction"},
    { topic: "well ordering", type: "Introduction"}
  ] } '   />
```

Figure 4.7: The structure of a chapter is defined in the HTML of its contents page. The code defining the structure of the chapter is shown in bold; the surrounding plain text code would be the same for any chapter.

The Wiki's display engine could even be altered to generate the hidden form from a tag so that the chapter author only has to enter a tag containing the topic structure. (It should be noted, though, that modern browser restrictions on cross-site scripting means that these chapter pages have to reside on the same server as the book). The links to the topics within the chapter are then Javascript calls to enter the chapter, rather than topic look up URLs. An example is given in Figure 4.8.

```
<a href="javascript: enterChapter(1);"
>A link to the first topic in the chapter</a>
```

Figure 4.8: Links into the chapter are Javascript calls rather than topic look up URLs.

A chapter bar, the upper of the two toolbars shown in Figure 4.2, allows the student to navigate between the topics in a chapter and to move back up the hierarchy of chapters that they have entered. The toolbar is implemented by splitting the page into a toolbar frame and a content frame. The frameset (the parent of both frames) contains Javascript that maintains the reader's navigational context within the chapters. This ensures that if the reader is reading a chapter, and encounters another page of type *chapter* within it, then this second chapter is treated as a subchapter (or subsection) of the first.

4.4 Closing Note

The content model described in this chapter is designed in a very open ended manner. The selection scripts could work on an arbitrary set of RDF tags, and using an arbitrary student model. For example, the scripts could look at how similar students reacted to content marked in similar ways in the RDF (perhaps students with very high scores prefer Prof. Smith's conciser entries instead of Prof. Jones's more verbose ones). Or we could monitor which entries actually caused students to improve in some skill.

Allowing the users of a book to add and alter content also raises questions about authenticity and quality assurance. Students could add incorrect explanations to the book. How, then, can a reader judge the relative merits of two different explanations? And how can the primary author of a book ensure that students are not misled by incorrect student-written explanations? The content model does provide two mechanisms to assist a book's primary author in this regard. He or she can tag content items so that they will always appear ahead of user-contributed items, and can also tag topics so that they will not accept any new contributions (see Section 4.2.1). One could, however, envisage extending the system to classify users according to their reliability. Reliability might be based on the users' roles (for example a teacher might be more reliable than a student), and also on analysis (for example students whose entries are often recommended by a teacher might be considered more reliable). The selection scripts might prefer entries from more reliable users, and less reliable users might be prevented from editing content that was written by someone deemed to be more reliable.

The usability experiment (described in Chapter 9), however, does not focus extensively on the content selection scripts themselves, but rather on the proof exercises that they are designed to support. For this reason, the selection scripts used in the usability experiment were kept deliberately naïve: they simply selected a page at random, weighted according to the $inbook$, $recommended$, and $rejected$ tags on the entry. Some discussion of details that might be appropriate to add to the student content model, and more complex page selection schemes, is given in Chapter 11.

CHAPTER 5

Server-Side Question Architecture

Figure 5.1 shows the architecture of an Intelligent Book question. The client components have been described in Chapter 3. This chapter describes how the server components support complex graphical questions, and how they allow different teaching pedagogies and modelling or reasoning systems to be used for different questions. The system I developed was written using Java Servlets, but the same structure could readily be re-implemented for other server platforms. The work described in this chapter forms part of a paper published in 2005 [BR05].

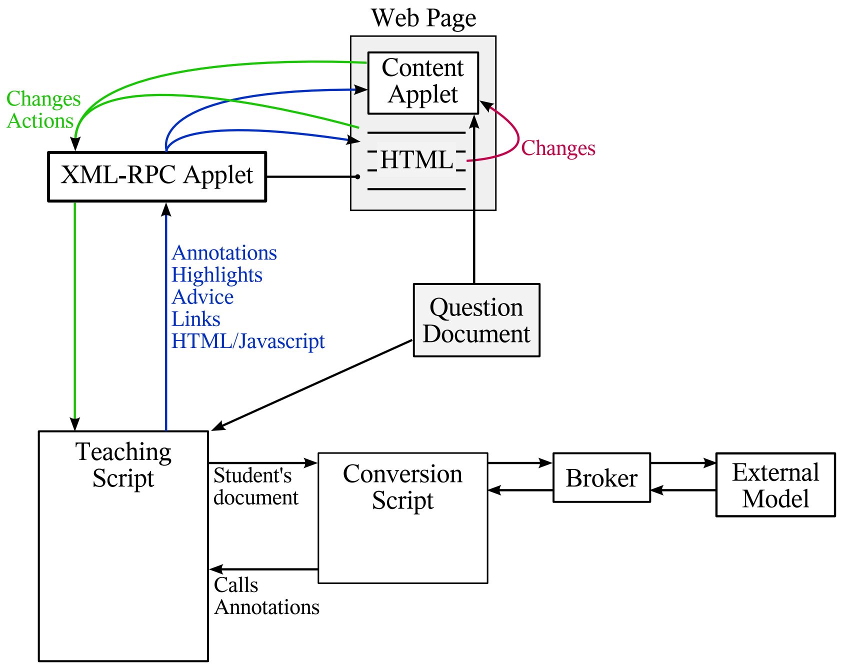


Figure 5.1: The architecture of an Intelligent Book question. The External Model represents any modelling or reasoning system that does not form part of the Teaching Script.

5.1 Teaching Scripts

5.1.1 Overview

Each question in an Intelligent Book is supported by a Teaching Script that describes how to respond to the student. Every XML-RPC call made by the client is a call to the Teaching Script. Every public method that an author writes into a Teaching Script can be called by the Web Page and the Content Applet, without requiring any alterations to the components in between.

Teaching Scripts are Java classes, so they support inheritance. A superclass provides the basic implementation of the features described in this chapter. Usually a subclass is defined for a particular kind of question, for example proof exercises, and that is subclassed again to provide the Teaching Scripts for individual questions. Teaching scripts can be written in any Java Virtual Machine supported language (including Jython and Groovy).

5.1.2 Supporting Different Teaching Pedagogies

In Chapter 1, I described how different teaching pedagogies could be appropriate for different questions. Recently, two successful automated tutoring techniques have been Model Tracing and Constraint Based Tutoring (see Sections 2.2.2 and 2.2.1). Model Tracing gives strategic feedback by comparing the students' actions to a set of production rules that represent an ideal solution strategy, and by looking for common procedural mistakes that students make. Constraint Based Tutors do not consider the students' actions directly, but examine what they produce at each step: the state of the answer. They check whether the state is valid or whether it breaks any domain rules. From this they assess the students' understanding of those rules. Judging the respective merits of the two techniques has been contentious [KWR05, MO06, KWR06]. However, a tutoring system can be likened to a complex state machine, and the difference between Constraint Based Tutoring and Model Tracing can be likened to the difference between a Moore machine and a Mealy machine. The Moore machine's (CBT's) output depends solely on the new state, while the Mealy machine's (Model Tracing's) output depends on the existing state and the student's input. For different questions, either kind of machine may be a more convenient representation, and it could sometimes be useful to follow a mixed approach. For example, a question might not contain a full set of Model Tracing rules, but only a few production rules to warn the student away from the most common procedural mistakes.

Any automated technique for providing advice or correction must involve some analysis using the old state, the new state, and a description of the change. The pedagogical methods differ by what kind of analysis model they use and where. So, Constraint Based Tutoring and Model Tracing each use Bayesian networks and a database of rules, in either a Mealy or Moore model of the system. Reactive Learning Environments use more specialised analysis models, usually using only the current state. For example, the electronics question in Chapter 3 used a set of state-based constraints plus *constraint propagation* in its analysis, but with no student model.

This gives us three steps to processing a student action, as shown in Figure 5.2. In the first step, the teaching script looks at the change the student is making and the current state of the student's work, and interprets what the student is trying to do. In the second step, the student's change is applied to the document in the database, giving a new current state of the problem. In the third step, the new state is examined.

This architecture considers the pedagogy within a question. Many techniques, for example

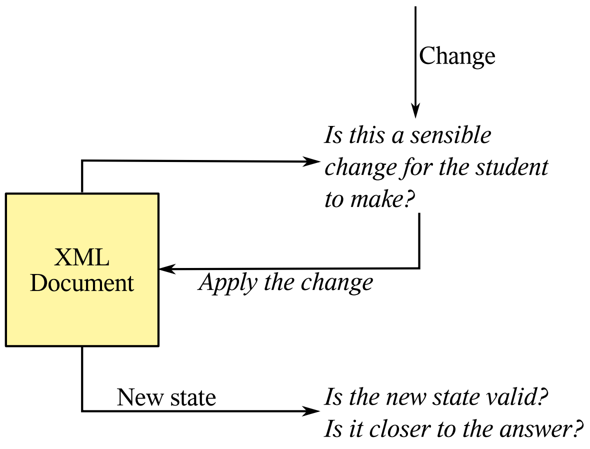

Change
Is this a sensible change for the student to make?
XML Document
Apply the change
New state
Is the new state valid?
Is it closer to the answer?

Figure 5.2: The teaching script breaks the analysis into a *state plus input* and a *new state* phase.

User-Adaptive Tests, also specify how the next question should be selected. There are two ways in which this can be supported. The first is by writing this into the Selection Scripts (described in Chapter 4) that choose an appropriate entry from the Book to show. The script would be made to examine the student model when considering which *exercise* should be chosen for the topic. The second is by altering the question. Because the question document is dynamically updatable, a User-Adaptive Test can be implemented in a single Teaching Script. As each question is completed, the Teaching Script would update the student's question document to show the next part of the test.

5.1.3 Advice Functions

In Chapter 4, I described how the content model is designed so that it can be extended by students but also referenced by the Teaching Scripts. However, when students ask for help with a topic, they are not always asking for content. They may be stuck in an exercise and be asking for analytical help examining how to solve a particular issue. It may be useful to include *ad hoc* analysis that relies on knowing how students are expected to answer the question. It would not always be possible, though, for the teacher to know whether this will be useful at the time the advice is written.

The system allows *advice functions* to be associated with topic keys in questions' Teaching Scripts. They are also associated with a *relevance function*. When the student asks, the Teaching Script will attempt to choose an advice function for the topic that is relevant to the current situation. If there is more than one relevant advice function, then the Teaching Script chooses a function based on which have been found to be useful in the past. (When the advice is presented, the student is asked to say whether or not it was helpful; these responses are recorded in the database along with the students' Question Documents.)

There can be two sets of advice functions. The first set take no arguments and can be called by the student at any time. The second set are for the student to ask for advice about comments made by the modelling or reasoning system. This set take a number of arguments to describe the context of the comment.

```
class ElectronicsTeachingScript extends TeachingScript

public void preChange() {
  // This script does no checking before the student's action is applied
}

public void postChange() {
  annotationsList = conversionProcessor.process(document, conversionScript);
  annotationsDoc.setContents(annotationsList);
  documentCache.put(annotationsDocumentKey, annotationsList)

  for(annotation in annotationsList) {
    if (annotation.type == consequentValue) {
      addResponse("content.setAttribute", annotation.path, "value", annotation.value);
      addResponse("content.setAttribute", annotation.Path, "setBy", annotation.setBy);
    }
    else if (annotation.type == contradiction) {
      addResponse("tutor.showContradiction", annotation);
    }
    else if (annotation.type == questionComplete) {
      addResponse("tutor.showSystemText", "Congratulations, you've finished the
          question");
    }
  }

}
```

Figure 5.3: Pseudocode for an Abstract Teaching Script class for the electronics question

5.1.4 A Hypothetical Example

The electronics question described in Chapter 3.2 predates the server architecture, but it is a suitably small example. The question gave students an electronic circuit, and asked them to choose appropriate values for currents, potentials, and component properties. A constraint propagator then worked out any other circuit values that followed logically by the rules of electronics, and also identified any contradictions.

Figure 5.3 shows Java-like pseudocode for an abstract teaching script describing this sort of question. It performs no checks before the student's change is committed. After the change has been applied to the document, the document is run through the conversion process to put the values into the constraint propagator. The output from this conversion process is collected as a list and stored in an annotation document. Each of these outputs might describe a consequent value that needs to be marked up at the client (or which might set a previously known value to *null*), a contradiction that needs to be explained to the student, or a message saying that the circuit is now fully specified.

Figure 5.4 shows a teaching script for a specific question. It sets the path and name of the question document in the database, and registers an *ad hoc* advice function suggesting that the user should start by setting I_c.

```
class Question1 extends ElectronicsTeachingScript

Constructor {
  super();
  adviceMap.add( {"advice_StartWithIc", "relevance_StartWithIc", "help"} );

  documentPath = "questions/electronics/MITquestions";
  documentName = "question1.xml";
}

public boolean relevance_StartWithIc() {
  // This advice is only valid if the student has not set the collector current
  setBy = evaluateXPath("//transistor/terminal[@name='C']/property[@name='I']/
      @setBy");
  return (! "student".equals(setBy) );
}

public void advice_StartWithIc() {
  addResponse("tutor.showSystemText", "Start by setting the collector current to a
      reasonable value.");
}
```

Figure 5.4: Pseudocode for a teaching script for the electronics question

5.1.5 Relationship to Servlets

A naïve approach in Java would be to make each Teaching Script a Servlet. However, the Java Servlet model expects Servlet objects to be thread-safe so that the same object can be used to handle many concurrent requests. While it is reasonable to expect experienced Java programmers to be able to write thread-safe servlets, this could pose a barrier to teaching staff who might not have as much specific experience with the Java Servlet platform (and it would be easy for script authors to forget this requirement if they did not work with the system regularly). For this reason, teaching scripts are not themselves Servlets but are disposable objects instantiated on a one-object-per-request basis by a central dispatching Servlet.

This central dispatching Servlet is registered in the Servlet container for all URLs matching the pattern `*.teachingScript`. It loads the target Teaching Script's class dynamically. To determine which class to instantiate, it uses the convention that the URL to call a Teaching Script must end "*`package.class`*`.teachingScript`". For example, a URL might end "`discreteMaths.fibonacci.FibMplusN.teachingScript`". (This also means that Teaching Scripts do not have to be registered with the Servlet container in `web.xml` as Servlets do.)

5.2 Conversion Scripts

Conversion Scripts are responsible for processing the student's document and presenting it to the External Model (any external modelling or reasoning system), if there is one, via the Broker. Sometimes, however, the Conversion Script contains its own modelling and there is no Broker or External Model. As described in Chapter 3, the student's question document is an XML document, so this involves converting its Document Object Model into a suitable form for the External Model.

Like XSLT [Cla99], the most common conversion technology applied to XML documents, Conversion Scripts work by associating patterns with templates. The pattern matches a part of the source document, and the template how to process that pattern. However, while XSLT is primarily designed to transform an XML document into another XML document, in an Intelligent Book we generally need to transform an XML document into a set of procedural actions. So, Conversion Scripts, rather than being written in an XML dialect, are written in Groovy [LCL+04] (a scripting language that interoperates well with Java) and the template is a Groovy closure of actions to perform. Conversion scripts are modular, in that they can include and extend other Conversion Scripts.

Figure 5.5 shows an extract from a Conversion Script for an informally modelled proof question (described in Chapter 10). The `proc.veracity` call registers the pattern. For this Conversion Script there are two lists of patterns – *veracity checkers* and *statement getters*. Veracity checkers know how to determine whether a particular piece of logic is true; statement getters know how to retrieve a statement ID from a piece of logic. The code in Figure 5.5 registers a veracity checker for a theorem. It says that to check whether the theorem is true, first the system should process the proof, and then it should process the theorem statement itself to see whether or not is has been shown to be true.

Figure 5.6 shows pseudocode for a Conversion Script for the electronics question.

5.3 Broker

A Broker is needed if the interface to the External Model is too complex or awkward to drive directly from the Conversion Script. For example, if the External Model is a separate process communicating over text streams, then a Broker must keep a pool of processes ready to handle requests. Once a Conversion Script has finished, the Broker resets the External Model for the next request rather than keeping its current state. This means that if the External Model

```
processor.veracity(MATHSTILES_NAMESPACE, "tile",
{it.getAttribute("definition")=="informalproofs:theorem"},
{
  processor.processVeracity(it, "mt:socket[@name='proofsteps']");

  processor.processVeracity(it, "mt:socket[@name='theorem']");
});
```

Figure 5.5: Registering a pattern in the Conversion Script for an informally modelled proof. The second (large) code closure describes the procedural actions to take for these tiles. `it` refers to the document element that has been matched by the pattern.

identifies an error, the Conversion Script should take action to collect any context information it needs (or serialise the External Model's state) before it exits. Otherwise when the student asks for advice, the Teaching Script would need to re-run the conversion process in order to analyse the External Model's state any further. The collected annotations are stored along with the student's document, so later calls to the Teaching Script can refer to them.

In the formally modelled proof exercises (Chapter 8), where the External Model is an automated proof assistant, the Conversion Scripts make frequent calls to write appropriately formatted proof commands into the Broker's buffer. When asked, the Broker writes the contents of the buffer to the proof assistant process and collects the responses. This happens regularly throughout the document, rather than only at the end. The responses are post-processed in the Broker, and passed back to the Conversion Script as annotations.

In the informally modelled proof exercises (Chapter 10), where the External Model is a Java

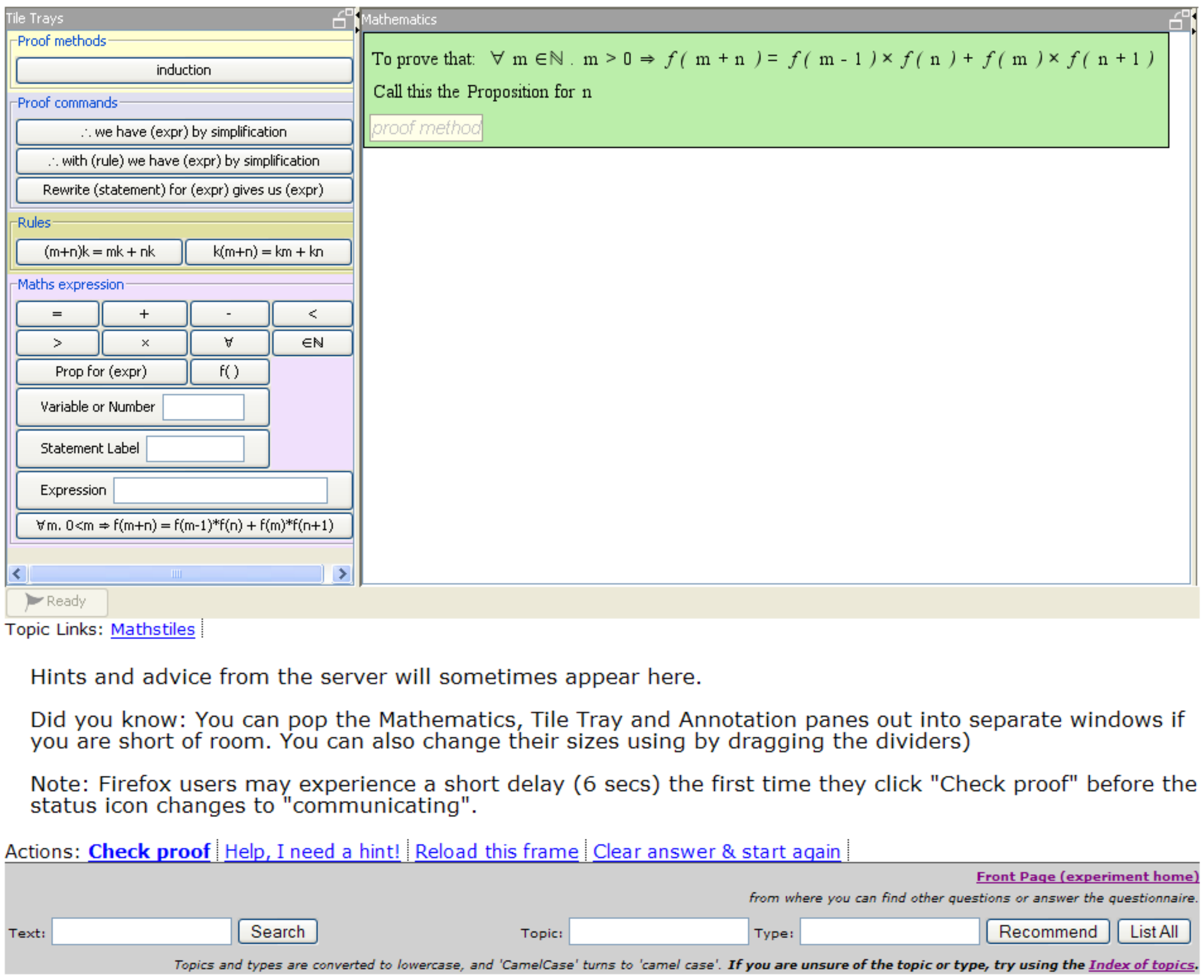
```
match mapping from a graphical notation path to an External Model path:
  register the mapping;

match component:
  process child elements;

match wire:
  process child elements;

match property:
  if ("student".equals(it.getAttribute("setBy"))) {
    mapping = map.get(getXPath(it));
    if (mapping != null) {
      Model.putValue(mapping, it.getAttribute(value));
    }
  }

match document:
  process all child elements;
  for (mapping in mappingsList) {
    annotations.add(new Annotation("consequentValue", Model.getValue(mapping),
        Model.getSetBy(mapping));
  }
  contradiction = Model.getContradiction();
  if (contradiction != null) {
    annotations.add(new Annotation("contradiction", contradiction));
  }
  if (Model.isFullySpecified()) {
    annotations.add(new Annotation("questionComplete"));
  }
```

Figure 5.6: Pseudocode for a conversion script for the electronics question

object written for the exercises, the interface is so simple that there is no need for a Broker.

5.4 Reusability of Components

If Content Applets and Brokers are written well, then Content Applets, Brokers, and External Models can often be reused across different kinds of question.



To prove that: $\forall\ m \in \mathbb{N}\ .\ m > 0 \Rightarrow f(\ m + n\ ) = f(\ m - 1\ ) \times f(\ n\ ) + f(\ m\ ) \times f(\ n + 1\ )$

Call this the Proposition for n

Topic Links: Mathstiles

Hints and advice from the server will sometimes appear here.

Did you know: You can pop the Mathematics, Tile Tray and Annotation panes out into separate windows if you are short of room. You can also change their sizes using by dragging the dividers)

Note: Firefox users may experience a short delay (6 secs) the first time they click "Check proof" before the status icon changes to "communicating".

Actions: **Check proof** | Help, I need a hint! | Reload this frame | Clear answer & start again

Front Page (experiment home)

from where you can find other questions or answer the questionnaire.

Text: Topic: Type:

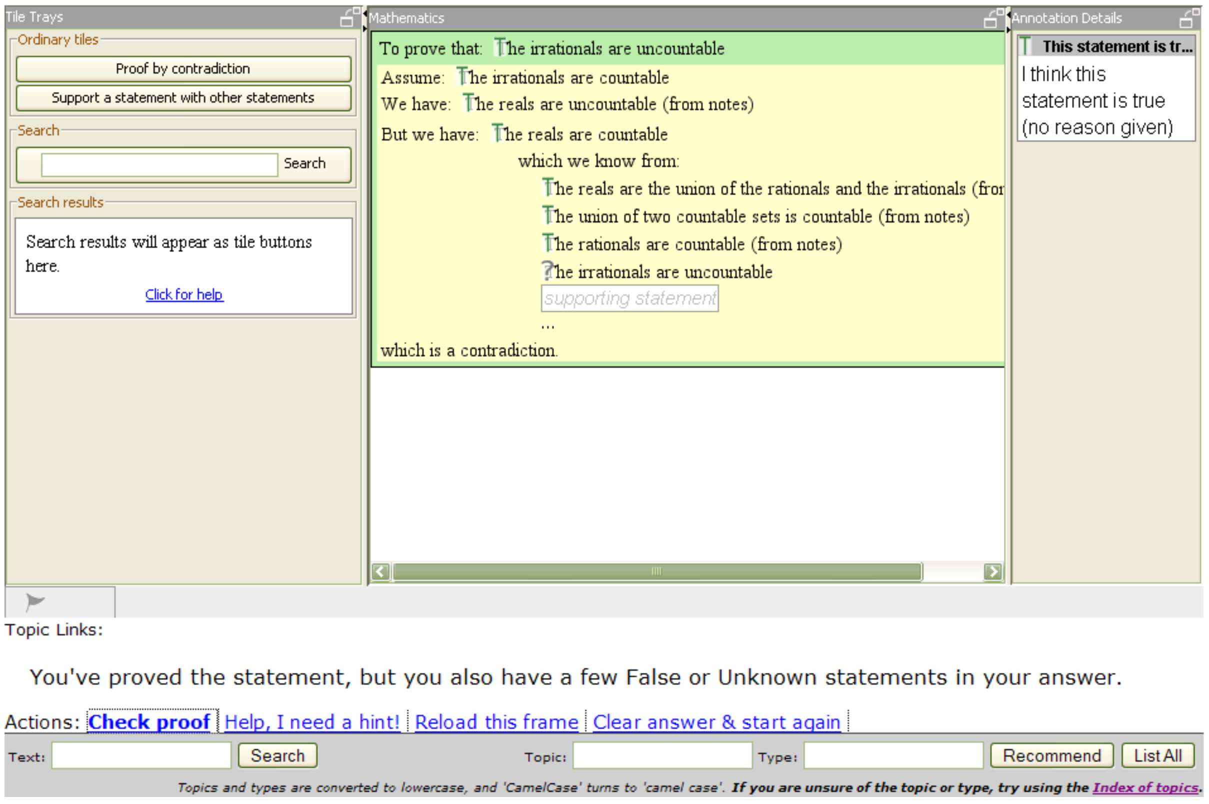
Topics and types are converted to lowercase, and 'CamelCase' turns to 'camel case'. ***If you are unsure of the topic or type, try using the Index of topics.***

Figure 5.7: An formally modelled question that uses MathsTiles and Isabelle/HOL.

Figure 5.7 shows a formal proof question that uses the Isabelle/HOL automated proof assistant as the model and a specialised interface called MathsTiles as the Content Applet. This type of question is described and developed in Chapters 6 to 9. Figure 5.8, shows a question that uses informal modelling rather than Isabelle/HOL. This uses a different Conversion Script (that includes its own modelling), but the same MathsTiles Content Applet. This question is developed in Chapter 10.

Figure 5.9 shows a proof exercise that uses Isabelle/HOL's native syntax. This uses a different Content Applet and Conversion Script, but the same Broker and External Model as the formal proof question in Figure 5.7.

In practice, the different Conversion Scripts tend to have a broadly similar structure (and Brokers, Teaching Scripts, and Content Applets similarly have their own common structures),


Tile Trays

Ordinary tiles

Proof by contradiction

Support a statement with other statements

Search

Search

Search results

Search results will appear as tile buttons here.

Click for help

Mathematics

To prove that: The irrationals are uncountable

Assume: The irrationals are countable

We have: The reals are uncountable (from notes)

But we have: The reals are countable

which we know from:

The reals are the union of the rationals and the irrationals (fro

The union of two countable sets is countable (from notes)

The rationals are countable (from notes)

?he irrationals are uncountable

supporting statement

...

which is a contradiction.

Annotation Details

This statement is tr...

I think this statement is true (no reason given)

Topic Links:

You've proved the statement, but you also have a few False or Unknown statements in your answer.

Actions: Check proof | Help, I need a hint! | Reload this frame | Clear answer & start again

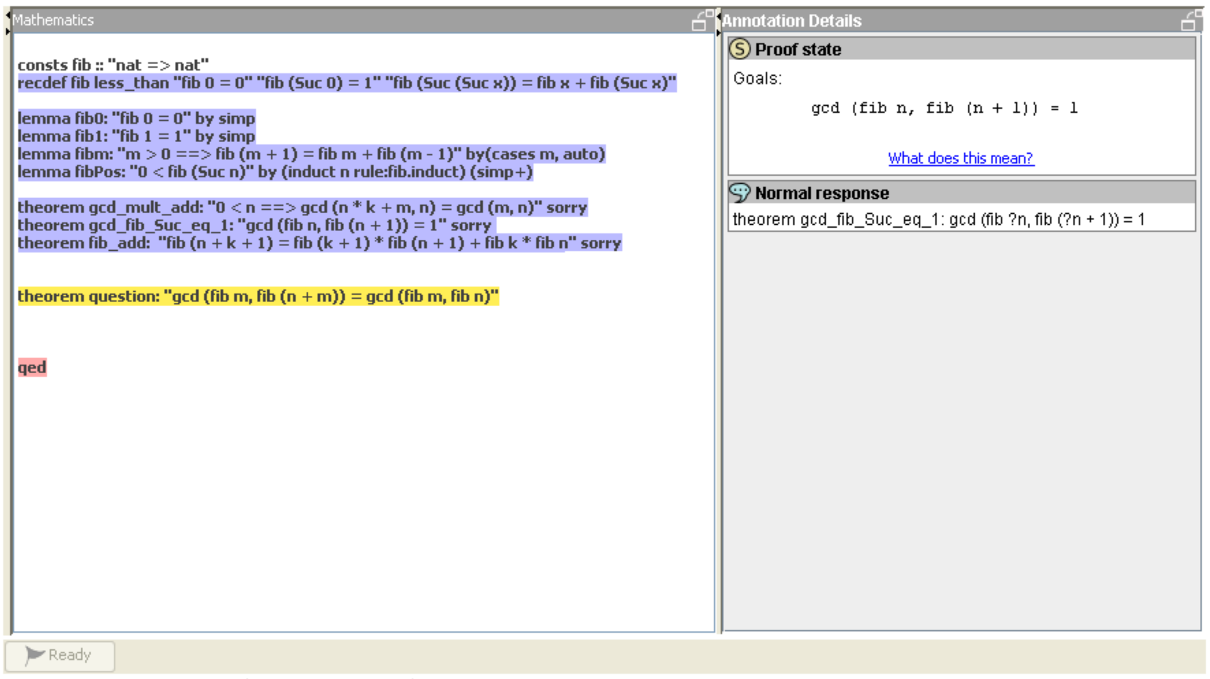
Text: Search Topic: Type: Recommend List All

Topics and types are converted to lowercase, and 'CamelCase' turns to 'camel case'. If you are unsure of the topic or type, try using the Index of topics.

Figure 5.8: An informally modelled question that uses MathsTiles but does not use Isabelle/HOL. (See Chapter 10 for further details about this question.)

so writing a new kind of question can be less effort than it might appear from Figure 5.1.

The electronics question described in Chapter 3 used the same question architecture at the client, but predates the server architecture. In early 2007, however, I re-implemented the electronics question using the server architecture described in this chapter.



```
consts fib :: "nat => nat"
recdef fib less_than "fib 0 = 0" "fib (Suc 0) = 1" "fib (Suc (Suc x)) = fib x + fib (Suc x)"

lemma fib0: "fib 0 = 0" by simp
lemma fib1: "fib 1 = 1" by simp
lemma fibm: "m > 0 ==> fib (m + 1) = fib m + fib (m - 1)" by(cases m, auto)
lemma fibPos: "0 < fib (Suc n)" by (induct n rule:fib.induct) (simp+)

theorem gcd_mult_add: "0 < n ==> gcd (n * k + m, n) = gcd (m, n)" sorry
theorem gcd_fib_Suc_eq_1: "gcd (fib n, fib (n + 1)) = 1" sorry
theorem fib_add:  "fib (n + k + 1) = fib (k + 1) * fib (n + 1) + fib k * fib n" sorry


theorem question: "gcd (fib m, fib (n + m)) = gcd (fib m, fib n)"



qed
```

```
Goals:

        gcd (fib n, fib (n + 1)) = 1
```

```
theorem gcd_fib_Suc_eq_1: gcd (fib ?n, fib (?n + 1)) = 1
```

Figure 5.9: A proof exercise that uses a native Isar Content Applet, but that uses the same Broker to talk to Isabelle/HOL.

CHAPTER 6

Proof Question Design Goals

In Chapters 3, 4, and 5, I described how I designed the Intelligent Book architecture so that it can support different kinds of question and different modelling or reasoning systems. A particular goal of this dissertation however, is to develop technologies for an Intelligent Book that can support proof exercises in introductory Number Theory for a first year Discrete Mathematics course.

This chapter describes the background and design goals for these proof exercises. Section 6.1 gives the background of the hypothesis and choice of model. Section 6.2 then describes three specific constraints on the proof exercises. The exercises are then developed and evaluated in Chapters 7, 8, and 9.

6.1 Background and Hypothesis

The first question that arises is *"what kind of model could be used to support proof exercises in introductory Number Theory?"* One possible choice is an automated proof assistant. These have been developed over many years to model and support the proofs of researchers and professionals. However, they are generally regarded to be difficult for novices to learn to use. From their experiences teaching postgraduates how to use the HOL system, Slind *et al* [SBC$^+$05] found interactive proof assistants to be "powerful but bewildering". They identified general reasons for this, including: "simply managing to formulate correct statements can be difficult"; "finding the correct tool to use at any point can be hard"; and "even remembering how to look for existing theorems to use can be hard".

Isabelle/HOL [NPW02], the proof assistant I use in this dissertation, is similarly complex. The two shortest introduction courses to Isabelle/HOL [Nip06, BK04], presented to research audiences, each take four sessions of 90 minutes and each include more than 300 slides. I asked informally on the Isabelle/HOL users' mailing list how long it might take a first year undergraduate to learn to use the system well enough to answer induction or case proofs on the Fibonacci sequence (using an example from the evaluation study in Chapter 9). The rough estimate I received from an experienced user was that if we offered a taught course in how to use the system then students "could do simple things within ten weeks" and "it might take as long as twenty weeks for an average student to become proficient at the level you are suggesting". I was also warned that if students could not already write a proof for a theorem on paper, they would not be able to prove it in Isabelle/HOL. In design-stage discussions with experienced users of HOL and Isabelle/HOL, I was also warned that the reasoning output of most proof assistants is very low level and would be difficult for students to follow.

My hypothesis, however, is that by using a very specialised interface to the proof exercises it is possible to provide something of educational value that students can learn to use much more quickly and with much less training. I have three reasons for believing that this might be the case:

- The interfaces of proof assistants appear, by and large, to have been optimised for experienced users who work with them regularly, rather than for novice users. There are many techniques in Human Computer Interaction research that can reduce the learning barrier for first time users – for example structured editing [TR81, AHW90] can help novices to work with a new syntax, but can be cumbersome for more experienced users [KU93].
- Answering a homework proof exercise is a different situation to attempting a proof in professional practice, because in a homework exercise the proof has been set by a teacher. The teacher has the opportunity to look at the question in advance and make alterations to ensure that an answer is achievable by students.
- Experience with the electronics question, described in Section 3.2, suggests that it is possible to relate automatically generated reasoning to a student's level of detail. In the electronics questions, initially the constraint propagator output explanations that (anecdotally) were too detailed and low level for students to understand. I found that a successful approach to solving this was to define the circuit diagram students would interact with separately from the TMS's constraint model of the circuit. The explanations were then automatically pruned so that only steps involving variables that were marked on the diagram were included. The principle here is that if the user interface is designed to represent the student's model of the question, then by mapping the reasoning onto that interface we are mapping it to a student's level of detail.

6.2 Design Goals

In this section, I describe three design goals for the exercises, and how those drove my design decisions. During development, I made minor compromises on the second and third goals, as described in Section 8.6, but nonetheless they were important to the design.

6.2.1 The exercises take place within a Web-based Intelligent Book

Providing proof exercises within a Web-based book places some extra limitations on the design. For example, the client component that the student works with must fit within a Web browser and be small enough to be downloaded over a slow connection. Also, the manner in which the student writes mathematics is limited to the mouse and keyboard, which traditionally are not an ideal mechanism for writing two-dimensional mathematical syntax. An Intelligent Book cannot, for instance, rely on recognising expressions written with a stylus because not all students will have one.

6.2.2 The student, not the system, should write the proof

Many proof assistants do not ask the user to write each line of the proof. Instead the user works by asking the assistant to apply tactics to statements on a goal stack. These tactics eliminate goals or produce new goal statements, until all the goals have been proved. It would be tempting, from a Human Computer Interaction perspective, to use a similar mechanism in the exercises. This way the student would not have to learn the prover's expression syntax (not even for the overall proof goal, which would be set by the question) but could focus on applying the appropriate tactics. However, this would also enable students to "game the system" by rapidly trying each tactic in turn, rather than actively thinking about the problem. This behaviour has been observed in a number of educational settings and correlates strongly with reduced learning outcomes [BCKW04].

Instead, I decided students should write the statements and expressions for each line of their proofs, as they do when answering proof exercises on paper, rather than have them generated by the system. This means that to use a tactic, the student has to think about what it will produce. So, the students' investment at each step is much greater and there is less scope for gaming the system.

For this reason, I chose Isabelle/HOL to act as the model. Its Isar proof language [Wen99, Nip03] supports "declarative" proofs that are somewhat similar to written proofs, rather than only supporting tactic scripts.

6.2.3 Proofs should resemble what students write on paper

While structured and menu-based editors have been known to reduce the burden of learning a new syntax (keywords and syntax rules can be recognised rather than recalled), this alone is unlikely to make Isabelle/HOL approachable for students with no experience of programming or proof. Isabelle/HOL contains both an inner HOL syntax and an outer Isar syntax. The outer Isar syntax contains keywords that appear similar in meaning but have very different effects. Fox example, the difference between the keywords `hence`, `thus`, `then`, `also`, and `moreover` is not readily apparent from the words themselves. There are also occasions where the same concept can be applied either at the Isabelle level or at the HOL level, for example whether the mathematical declaration *for all* is made using `!!` or `ALL`, and this decision will affect later proof commands.

Also, as described in Section 6.1, the user interface should represent a "students' model" of the question rather than the reasoning system's model. In this case, I decided that the statements students make in questions should more closely resemble statements they might make on paper, rather than mimicking the Isar language exactly. (That is not to say, however, that they look identical to written proofs.)

A related point is that when students write proofs on paper, they do not always take the strictly top-down approach that traditional structured editors encourage. The interface should not force them into that approach. As an example, it would be very unusual for a student writing an algebraic expression on paper to write the symbols in the hierarchical order of the expression's parse tree. Students may wish to start in the middle of the expression, or may wish to sketch out parts of the expression and then link them up. The interface should make some attempt to support this.

CHAPTER 7

MathsTiles

Students using an Intelligent Book should be able to work with notations that are appropriate to the subject matter. An Intelligent Book for Discrete Mathematics, then, needs a way for students to work with mathematics and proofs. This chapter describes the interface that I developed for this purpose. It has been described briefly in a 2005 paper [BR05] and more fully in a paper to appear in 2007 [BR07].

7.1 Overview

MathsTiles is an interface for students to edit structured content, such as mathematical equations and proofs, within Web pages. The syntax is not fixed but is configurable from question to question. This means that MathsTiles is not itself a formal or semantic language for mathematics, but is a structured interaction language designed so that the constructed mathematics can easily be transformed into other representations (including formal and semantic representations). For example in the proof exercises, tiles gain a semantic meaning on the server because they are transformed into Isabelle/HOL's modelling language as described in Chapter 8.

MathsTiles is designed to achieve the following goals:

1. **Resemblance to maths**.
   The notation used to enter and manipulate maths should look like the notation students are expected to write on paper, for example in their exam answers. If the notation were very different, for example a text-based formula language, then this would add a learning burden which is not directly related to the material being taught.

2. **Ease of alteration**.
   Students can be expected to enter incorrect expressions and proofs much of the time – if they already knew the material they wouldn't be students. So, it is important that students can make changes to their expressions easily.

3. **No forced order**.
   The interface should not force the student to write syntax in a particular sequence. While

there are occasions where teachers might want students to use a particular methodology, this should be enforced in the explicit teaching feedback, rather than as an implicit by-product of the interface design. So for example, students should be able to build the middle parts of an expression before the outer parts if they want to.

4. **Low commitment**.
   Students should be able to write and play around with fragments of answers without being committed to them. The interface should allow students to construct and examine as many answer fragments as they like in parallel.

5. **Progressive evaluation**.
   Sometimes, students might know what part of the proof or expression needs to look like, but get stuck on how to complete the structure. They should be able to ask for feedback from the tutor on an incomplete answer fragment.

6. **Ease of authoring**.
   Because it is not possible to identify in advance all the different pieces of mathematics (which includes proof structures and arguments as well as symbols) question authors will wish to include in their questions, it needs to be simple for authors to implement new pieces of notation.

7. **Reasonable size for the Web**.
   While fast broadband connections are becoming more common, performance over slower or more congested networks should still be reasonable. This means that both the code size of the client applet and the size of the MathsTiles documents need to be reasonably small.

Tiles containing arbitrary pieces of maths can be added to the page, dragged around and dropped into sockets in other tiles to build up the structure of an expression or proof by containment. In this way, the notation is kept closely mapped to handwritten mathematics, but the students are exposed to the hierarchical nature of the expressions they are building. A simple example of some tiles is shown in Figure 7.1.

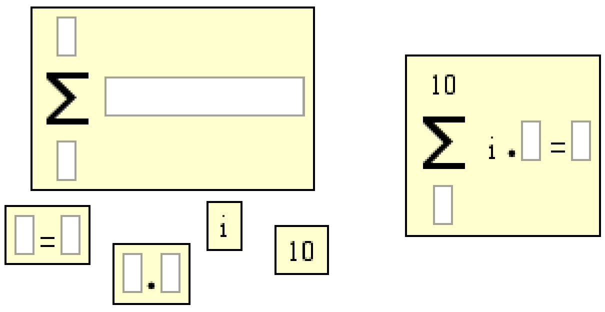


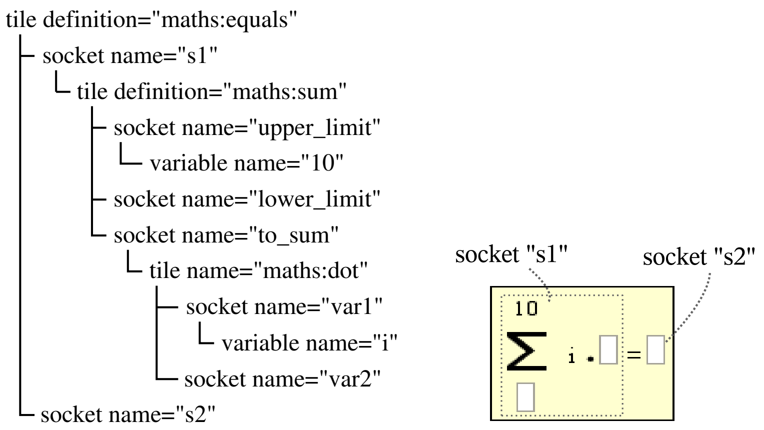
Figure 7.1: Some maths tiles, loose and combined

Tiles can be pulled out of and dropped into sockets by holding the `Ctrl` key when pressing or releasing the mouse over the tile or socket, so the effort required to change a structure is low. When a student drops a tile into a socket in another tile, the border of the contained tile is removed so that the appearance of the constructed maths is not interrupted. However, the tile border reappears when the mouse is moved over the tile, giving the student a clear sense of the structure of the tile.

The fact that tiles and groups of tiles can sit on the page without being combined into the student's answer means that students are able to write parts of their answer without being committed to them. New parts of an expression or proof can be plugged in without discarding the old parts. Because tiles can be combined as easily in an outside-in or inside-out order, the student is not constrained to working in a top-down or bottom-up manner.

7.2 Document Structure

As described in Chapters 3 and 5, the student's document is an XML file and its Document Object Model is updated in real-time on both the client and the server as the student works on it.



```
tile definition="maths:equals"
├ socket name="s1"
│  └ tile definition="maths:sum"
│     ├ socket name="upper_limit"
│     │  └ variable name="10"
│     ├ socket name="lower_limit"
│     └ socket name="to_sum"
│        └ tile name="maths:dot"
│           ├ socket name="var1"
│           │  └ variable name="i"
│           └ socket name="var2"
└ socket name="s2"
```

Figure 7.2: The combined tiles from Figure 7.1, together with the XML of the structure, shown as a tree. The sockets of the `equals` tile have been labelled on the diagram.

Figure 7.2 shows the combined tiles from Figure 7.1 together with their XML structure. The outermost `tile` element has its `definition` attribute set to `maths:equals`. Most tiles in a document, like this one, are *defined tiles*. Their appearance and structure are not fixed in the MathsTiles program, but are described by *tile definitions*. Here, the tile is defined by the `equals` tile definition in a separate tile document called `maths`.

Within the `tile` element are two `socket` elements which are the two sockets of the `equals` tile. The socket called `s2` (the right socket) is empty, while the socket called `s1` (the left socket) contains a `sum` tile. This `sum` tile in turn contains sockets, some of which contain other tiles. Note that the socket names are local to the tile – if there was a second `equals` tile on the page, its left and right sockets would also be named `s1` and `s2`.

Figure 7.3 shows the tile definition for the `sum` tile in Figure 7.1. Within the `tileDefinition` element, there are three `socketDefinition` elements that define the three sockets in `sum` tiles. The names of the tile's sockets in Figure 7.1 match the names of the socket definitions in Figure 7.3. Here, the socket definitions have specified the sockets' widths and heights. There is also a `text` element that defines the sum symbol that appears on the tile.

The `layout` element corresponds to the fact that the tile definition's `layout` attribute is set to `InstructionLayout`. This `layout` element contains a sequence of `move` and `pull` elements that describe operations that will arrange the sockets and text on the tile appropriately. Alternatively, if the layout attribute was set to `BaselineFlowLayout`, then all the components of the tile would be arranged left to right, vertically aligned by the baselines of any text that

```
tileDefinition name="sum" layout="InstructionLayout"
├ socketDefinition name="to_sum" width="100" height="20"
├ text name="sum_sign" font-name="Math" font-size="20"
│ └ Σ
├ socketDefinition name="upper_limit" width="10" height="20"
├ socketDefinition name="lower_limit" width="10" height="20"
└ layout
  ├ move c1="sum_sign" e1="North" by="5" c2="upper_limit" e2="South"
  ├ move c1="to_sum" e1="West" by=">5" c2="sum_sign" e2="East"
  ├ move c1="to_sum" e1="v_middle" by="0" c2="sum_sign" e2="v_middle"
  ├ move c1="lower_limit" e1="North" by="5" c2="sum_sign" e2="South"
  ├ move c1="lower_limit" e1="h_middle" by="0" c2="sum_sign" e2="h_middle"
  └ move c1="upper_limit" e1="h_middle" by="0" c2="sum_sign" e2="h_middle"
```

Figure 7.3: The definition and layout of a Sum tile. "Component" and "Edge" have been abbreviated to "c" and "e" in this figure. The horizontal middle, vertical middle, and text baseline are also edges that can be used in alignment operations.

appears on them. (The baseline of a tile laid out using `InstructionLayout` is the baseline of the first element in its tile definition.)

A tile is loosely coupled to its definition, so the visual appearance of a MathsTiles document can be changed by loading it with a different set of tile definitions. This is not as flexible as a stylesheet, however, because changing a tile definition always changes the appearance of every tile in the document referring to it.

7.3 Definable Tile Components

Tile definitions can include the following components:

Text.
The text that appears on a tile is specified by `text` elements in the tile definition. By setting the `visible` attribute to an XPath [CD99] expression, a piece of text can be made to appear only if the expression evaluates to $true$. This can be used, for example, to make the brackets on a `plus` tile to only appear when the tile is within a socket in a `times` or `power` tile, as shown in Figure 7.4.

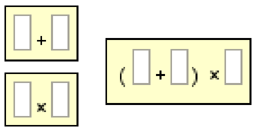


Figure 7.4: The `visibility` attribute of the brackets is set so that they will only appear when the `plus` tile is inserted into a socket in a higher priority tile. (For the same reason as the limitation on the type system, as described in Section 7.7, the higher priority tiles are listed explicitly in the expression.)

Symbol.
Symbols can be defined using the Scalable Vector Graphics (SVG) path syntax [FFJ03], and given a name. Once defined, a symbol can be placed on a tile by including a `symbol`

element in the tile's definition. As with text, each symbol on a tile can be given a visibility that depends on an expression.

Socket.
Each socket is defined by a `socketDefinition` element in the tile definition. Background text can be set to appear on the socket when it is empty. The colour, height, and width of the socket can also be specified.

The `tagName` attribute provides a rudimentary way of setting what kinds of tile can be inserted into the socket. If it is set then only tiles whose element tag (for non-defined tiles) or definition (for defined tiles) appears in the list of names in the `tagName` attribute will be accepted into the socket. When a tile is being moved and the `Ctrl` key is pressed, the socket underneath the tile that the student might want to drop it into will outline itself in green or red depending on whether it would accept the tile or not. As discussed in Section 7.7, this is only a rudimentary substitute for a type system, and if an author wrote further tiles it might be necessary to alter the `tagName` attributes on the sockets of existing tiles.

Socket List.
Horizontally or vertically arranged lists of sockets can also be placed on a tile. Socket lists can have a specified number of sockets, or they can be set to expand automatically so that there is always an empty socket in the list. Expanding socket lists place an ellipsis ('`...`') at the end of the list to show that it will expand. A `socketDefinition` within the `socketListDefinition` defines what the sockets in the list should look like.

Three attributes of tiles are also worth noting. `Selectable` (default *yes*) sets whether or not the user can select this tile. Unselectable tiles are effectively stuck on the page or in their sockets. If they are stuck within sockets then the socket border will not highlight when the mouse moves over the tile, and the unselectable tile will appear to be an integral part of its parent tile. `Delible` (default *yes*) sets whether or not the tile can be deleted. `Background` sets the background colour of the tile.

7.4 Inheritable Attributes

Some attributes can be inherited from the parent socket or tile. The rules of inheritance are that if an attribute is not set on an element, then first the corresponding definition element is checked (`tileDefinition` for `tile`, etc). If the definition element does not set the attribute, then the parent element is checked. The inheritable attributes include: `selectable`, `background`, `socketBackground`, `delible`, `font-size`, `font-style`, and `font`.

For numeric attributes, particularly `font-size`, if the attribute value begins with a '`*`', then system will attempt to set it to the inherited value multiplied by the number after the '`*`'. This allows, for example, the text on an expression to be scaled down if the expression is placed into a socket that represents a subscript.

Attributes can be reset (set to nothing) by setting them to an invalid value.

7.5 Non-Defined Tiles

In addition to defined tiles, MathsTiles also provides four hardcoded kinds of tile for convenience with mathematics:

Variable.
A variable is a simple tile containing text that matches its `name` tag. It is also useful for representing numbers.

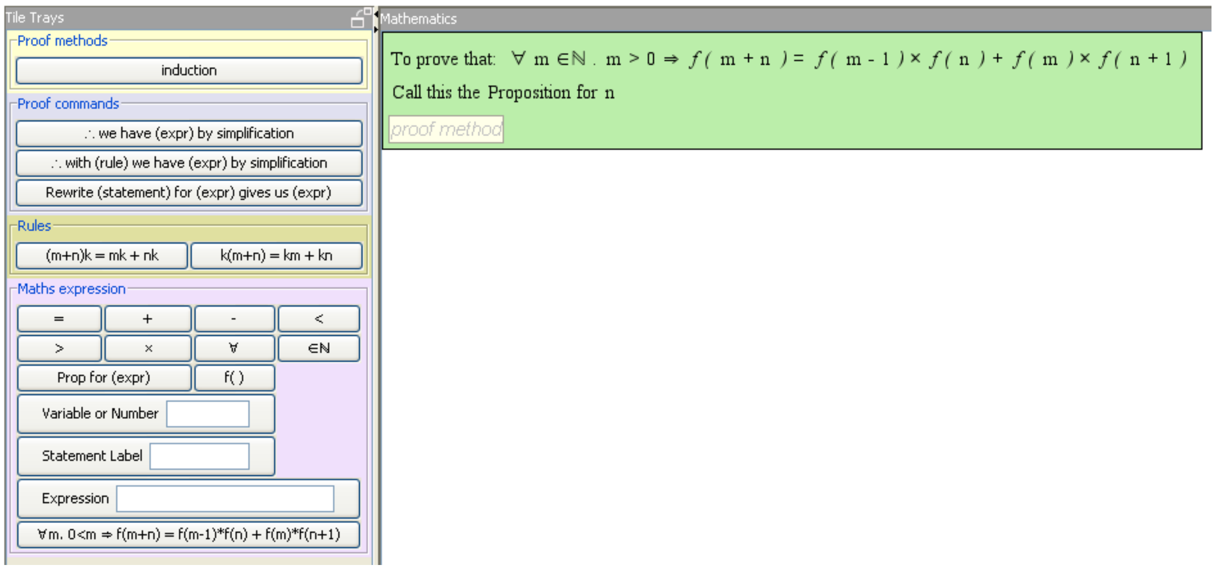
Function.
A function contains text that matches its `name` tag, and sockets for its parameters. The sockets are surrounded by parentheses. Functions can take a configurable number of parameters, or can be set to automatically expand. A separator character can also be configured.

Labelled Statement.
A labelled statement is a tile that contains a socket for the statement, and text for the label. The label is set using the `id` attribute.

Statement Reference.
A statement reference is a simple tile containing text that matches the label of the statement it references (defined by the `id` attribute).

7.6 Tile Trays

The set of buttons and controls that a student can use to add tiles to a proof (called a *tile tray*) is also defined in XML. It can form part of the student's proof document, or it can be part of a separate document in the same way that the tile library documents are. In Figure 7.5, the tile tray is to the left of the picture.



To prove that: $\forall\ m \in \mathbb{N}\ .\ m > 0 \Rightarrow f(\ m + n\ ) = f(\ m - 1\ ) \times f(\ n\ ) + f(\ m\ ) \times f(\ n + 1\ )$

Call this the Proposition for n

Figure 7.5: The set of controls that the student can use to add tiles to the page is also configurable in XML.

The controls that can be placed into the tile tray are:

TileButton.
Inserts a single tile, as specified by the definition referred to by the `definition` attribute.

XmlButton.
Rather than inserting a single tile, an XML Button inserts tiles to match a defined XML structure. This is useful to provide both for commonly-used expressions (such as the expression contained in the theorem to be proved), and also to insert a nest of tiles but treat it as a single tile. Marking the contained tiles in the nest as unselectable in the XML prevents them from being pulled out of their parent tile.

VariableButton.
Inserts a variable. The name is specified by typing it into an edit box set into the button.

StatementButton.
Inserts a Statement Label or a Statement Reference. If the text typed into the edit box (within the button) is already the label of a statement on the page, then a Statement Reference is added. If not, then a Statement Label is added. If the edit box is left blank, then the button automatically generates a new label.

Tabbed Pane.
Holds a set of tabs.

Tab
A labelled tab group that can hold a set of buttons. (May or may not be within a Tabbed Pane.)

Expression Button.
Parses an expression typed or pasted in by the user, and produces a tile structure to match that expression. It's primary purpose is that if a hint message or a response from the prover contains an expression, the user should be able to paste that expression into the proof. It is also included, however, because simple one-dimensional expressions such as $3 + 4$ are much faster to type than to construct with the mouse. (See Section 7.7.)

Tile Search Button.
This takes advantage of the dynamic nature of the tile tray. The tile tray, like the proof document, can be altered at run-time by scripting calls from the server. This means that not all of the buttons the student will use for the question need to be in the tile tray at the start. The `TileSearchButton` sends the student's search query to a function in the question's teaching script, which usually responds by adding found tile buttons to a "search results" tab in the tile tray.

7.7 Future Work

This section describes two possible extensions that have been omitted in order to keep the MathsTiles applet down to a reasonable size and to keep the interface straightforward for the evaluation.

Despite the fact that being able to type is known to be useful in structured editors, it is not possible to edit in MathsTiles by typing. The only expression control that there is uses an expression parser that only accepts a few formats (XML, Isabelle/HOL expressions, and basic arithmetic), although it is reasonably forgiving of errors. The reason for this omission becomes apparent when you consider that MathsTiles does not have a fixed syntax, but a changeable syntax from question to question. It is also technically possible for new tile definitions to be introduced during questions. Furthermore, many of the tiles use a two-dimensional syntax ordered by layout rules. It is not obvious what is the most usable technique to convert from a one-dimensional syntax (text) to an *ad hoc* two-dimensional syntax. So, this is left for future work.

It would be useful if parts of a tile or socket definition could depend upon an attribute of the tile or socket. For example, if a piece of text that appears on a defined tile could be set to match the `name` attribute of the tile, then it would not have been necessary to hardcode variable and function tiles. As a second example, if a socket could be defined to only accept tiles where an expression such as "$socketDefinition.type = tile.type$" was true then this would allow question authors, if they wished, to prevent students from inserting tiles into unsuitable sockets. Currently the `tagName` attribute provides only rudimentary support for this. However, MathsTiles was designed to work with version 5.0 of the Java Runtime Environment, which includes an expression parser for XPath but not for any more general purpose languages. XPath expressions cannot bridge documents and we usually keep the tile definitions in library documents that are separate from the question document. So, we would need to include our own general purpose expression language for tiles, which we decided would make the applet size too large. Java version 6.0 does include general purpose languages that could be used for this purpose in future versions.

7.8 Conclusion

Although structured editing is a well established technique, as discussed in Section 2.4.3, there are a number of aspects in which MathsTiles is unique.

Allowing tiles to be scattered on the page makes it simpler to work in a bottom-up manner than in many structure based editors, and many answer fragments may exist simultaneously. Whereas in most programming languages, code needs to be commented out or cut and paste into a notepad to detach it from the program without deleting it, a MathsTiles structure can simply be unplugged from its parent and left on the page.

The ability to define and configure new kind of tiles allows MathsTiles to be adapted to very different kinds of question – for example the formal proof exercises in Chapter 8 versus the informal proof exercises in Chapter 10. The informal proof exercises also take advantage of the fact that the document, the tile tray, and the library definitions can all be updated from the server during a question using the API described in Chapter 3.

Thirdly, as described in the next chapter, the tile syntax does not need to directly match the underlying modelling language. This is both in terms of being able to translate syntax elements into different language, and also because tiles can be forced to stick together by making some tiles unselectable. This allows the granularity of interaction (what kinds of structures are considered atomic) to be altered in places, rather than always using a keyword-level granularity.

CHAPTER 8

MathsTiles as a Proof Language

In Chapter 7, I described how MathsTiles works as a structured interaction language and an editor for redefinable mathematics. In this chapter, I describe how I have used MathsTiles to allow students to write proofs that can be translated automatically into Isabelle/HOL's Isar language in proof exercises. The proof exercises are introduced with a straightforward example before the principles behind the exercises are described.

8.1 A Straightforward Example

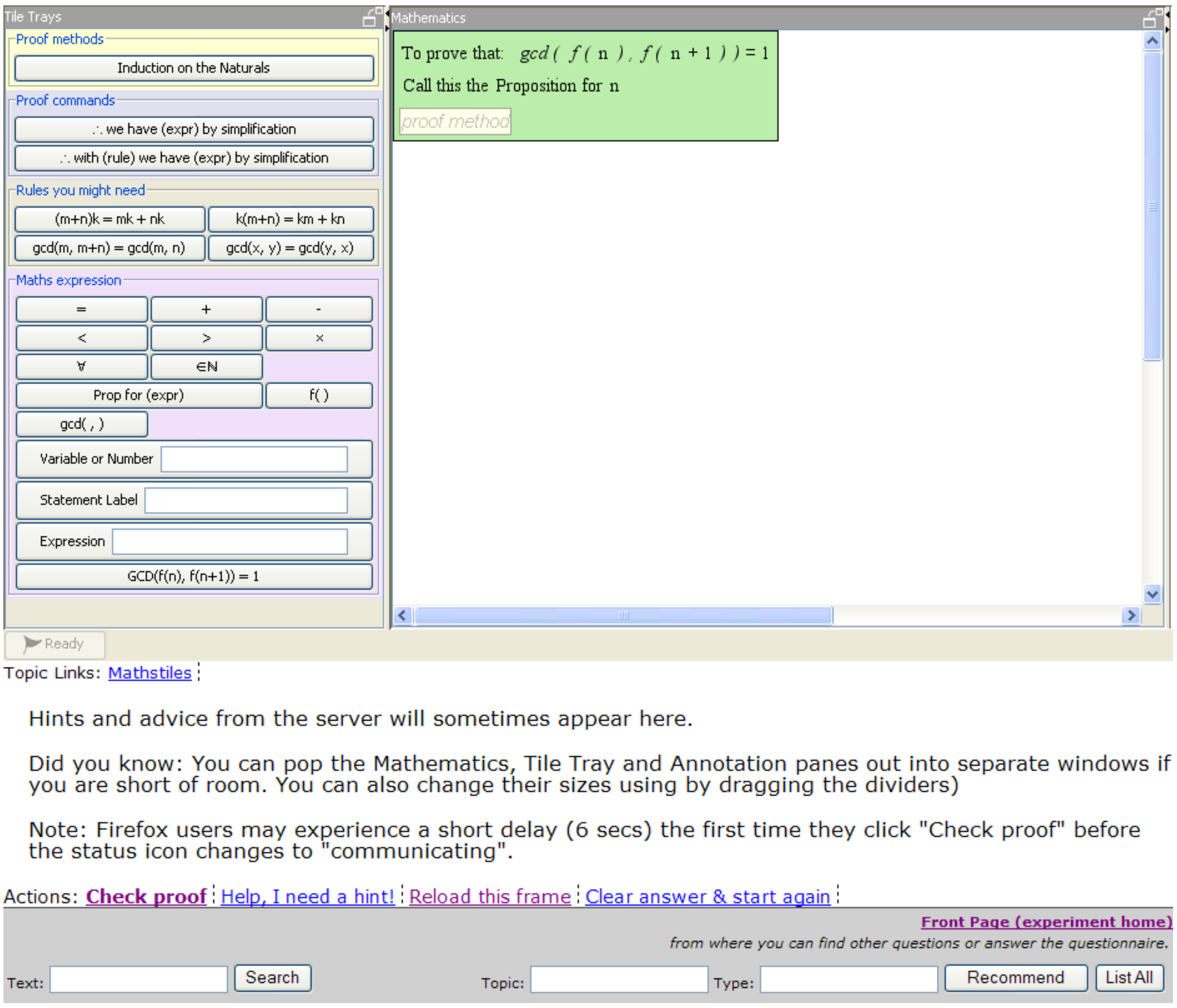
This is an example of a typical proof exercise using the system. The question is a homework exercise from the lecture notes of the first year undergraduate Discrete Mathematics course in the Computer Science tripos.

Students are given the following definitions:

> The Fibonacci sequence is defined as:
> $f(0) = 0$
> $f(1) = 1$
> $f(n+2) = f(n) + f(n+1)$
> where $f(n)$ represents the nth Fibonacci number.
>
> The Greatest Common Divisor is defined as:
> $GCD(0,0) = 0$
> $GCD(a,0) = a$
> $GCD(0,b) = b$
> $GCD(a,b)$ is the largest natural number that divides both a and b without leaving a remainder.

They are then asked to prove by induction that $GCD(f(n), f(n+1)) = 1$.

Initially, the question appears as shown in Figure 8.1. (The definitions are not shown in the figure, but are above the exercise in the Web page.) The tiles on the page at the start of a



To prove that: $gcd(f(n), f(n+1)) = 1$

Call this the Proposition for n

Hints and advice from the server will sometimes appear here.

Did you know: You can pop the Mathematics, Tile Tray and Annotation panes out into separate windows if you are short of room. You can also change their sizes using by dragging the dividers)

Note: Firefox users may experience a short delay (6 secs) the first time they click "Check proof" before the status icon changes to "communicating".

from where you can find other questions or answer the questionnaire.

Figure 8.1: An induction proof question waiting to be filled in. Because this question is specifically set as an induction proof, no other proof methods are available.

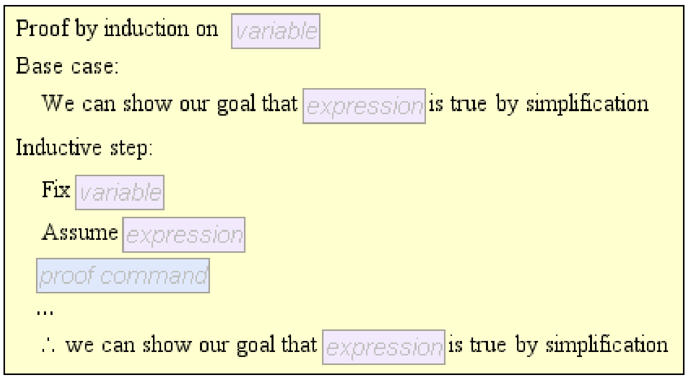
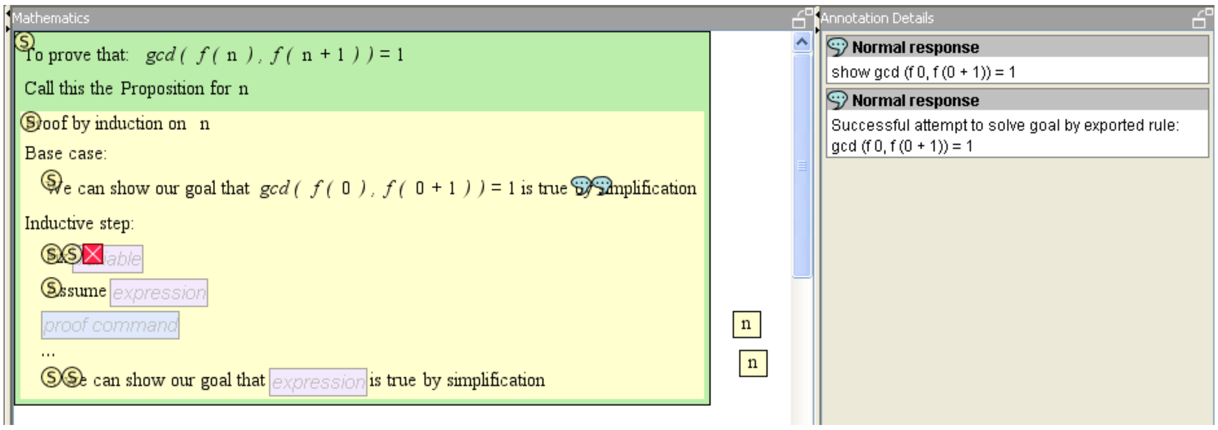

Proof by induction on variable
Base case:
We can show our goal that expression is true by simplification
Inductive step:
Fix variable
Assume expression
proof command
...
∴ we can show our goal that expression is true by simplification

Figure 8.2: The induction tile for the worked example.

question are fixed in place and coloured green; these need to be filled out to complete the proof. The only socket available in the answer asks for a proof method. In the tile tray, there is only one button in the section marked "proof methods": induction. The question specifically asks the student to use induction, so no other methods are allowed. The induction tile for this question is shown in Figure 8.2.

The induction tile has a number of sockets to fill: the induction variable, the goal statement for the base case, and several sockets in the inductive step. Let us induct on n. Now let us consider the goal for the base case. At the foot of the tile tray in Figure 8.1, there is a button that will insert the entire expression $GCD(f(n), f(n+1)) = 1$, which is the statement to be proved. For the base case, we must show that this statement is true where $n = 0$. So, let us insert this expression into the goal and substitute 0 tiles for the n tiles. Filling these in and clicking "*Check Proof*" we find that the base case can be solved by the simplifier, as shown in Figure 8.3.

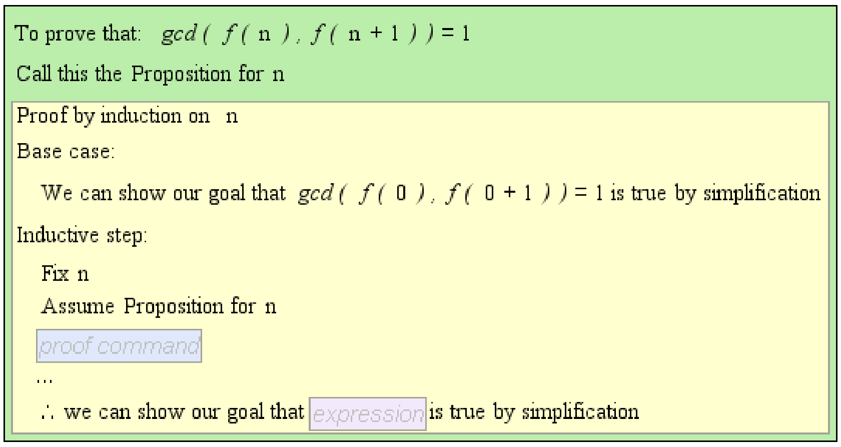


To prove that: gcd (f (n), f (n + 1)) = 1
Call this the Proposition for n
Proof by induction on n
Base case:
We can show our goal that gcd (f (0), f (0 + 1)) = 1 is true by simplification
Inductive step:
iable
Assume expression
proof command
...
e can show our goal that expression is true by simplification
n
n

Normal response
show gcd (f 0, f (0 + 1)) = 1
Normal response
Successful attempt to solve goal by exported rule:
gcd (f 0, f (0 + 1)) = 1

Figure 8.3: The base case can be solved by the simplifier.

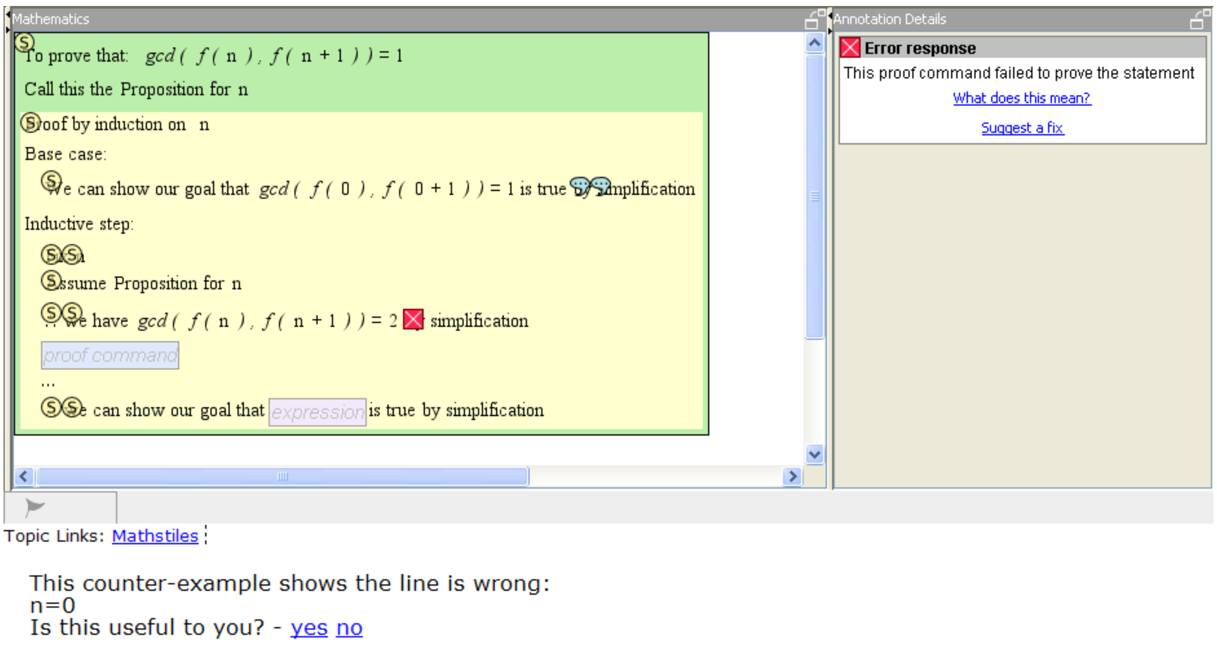
For the inductive step, we need to assume that the proposition is true for some arbitrary value. We achieve this by filling in the *Fix* and *Assume* tiles in the inductive step. Let us fix n. We could explicitly assume that $GCD(f(n), f(n+1)) = 1$, but here let us use the shortcut `Proposition for n`. Checking the proof again gives us the situation in Figure 8.4.



To prove that: *gcd* (*f* (n) , *f* (n + 1)) = 1
Call this the Proposition for n
Proof by induction on n
Base case:
We can show our goal that *gcd* (*f* (0) , *f* (0 + 1)) = 1 is true by simplification
Inductive step:
Fix n
Assume Proposition for n
proof command
...
∴ we can show our goal that expression is true by simplification

Figure 8.4: The question with the base case completed and the step assumption filled in.

To see what happens when we introduce an error into our proof, let us insert the statement "∵ *we have* $GCD(f(n), f(n+1)) = 2$ *by simplification*" into our script. The "*by simplification*" justification makes the system use a set of term rewriting rules to try to show that the statement is true (see Section 8.4). Clearly, however, the statement we have just added is not true because we earlier assumed that expression equals 1, not 2. The error this statement produces is "*This proof command failed to prove the statement*". Because this is an error, the annotation for it has a "*Suggest a fix*" link underneath it. In this case, when the link is clicked, the helper function on the teaching script that is called looks for a counter-example, trying the numbers from 0 to 20. Zero should be identified because $GCD(f(0), f(1)) = GCD(1, 1) = 1$. Figure 8.5 shows a screenshot of the returned counter-example.



To prove that: *gcd* (*f* (n) , *f* (n + 1)) = 1
Call this the Proposition for n
Proof by induction on n
Base case:
We can show our goal that *gcd* (*f* (0) , *f* (0 + 1)) = 1 is true by simplification
Inductive step:
Fix n
Assume Proposition for n
∵ we have *gcd* (*f* (n) , *f* (n + 1)) = 2 by simplification
proof command
...
∴ we can show our goal that expression is true by simplification

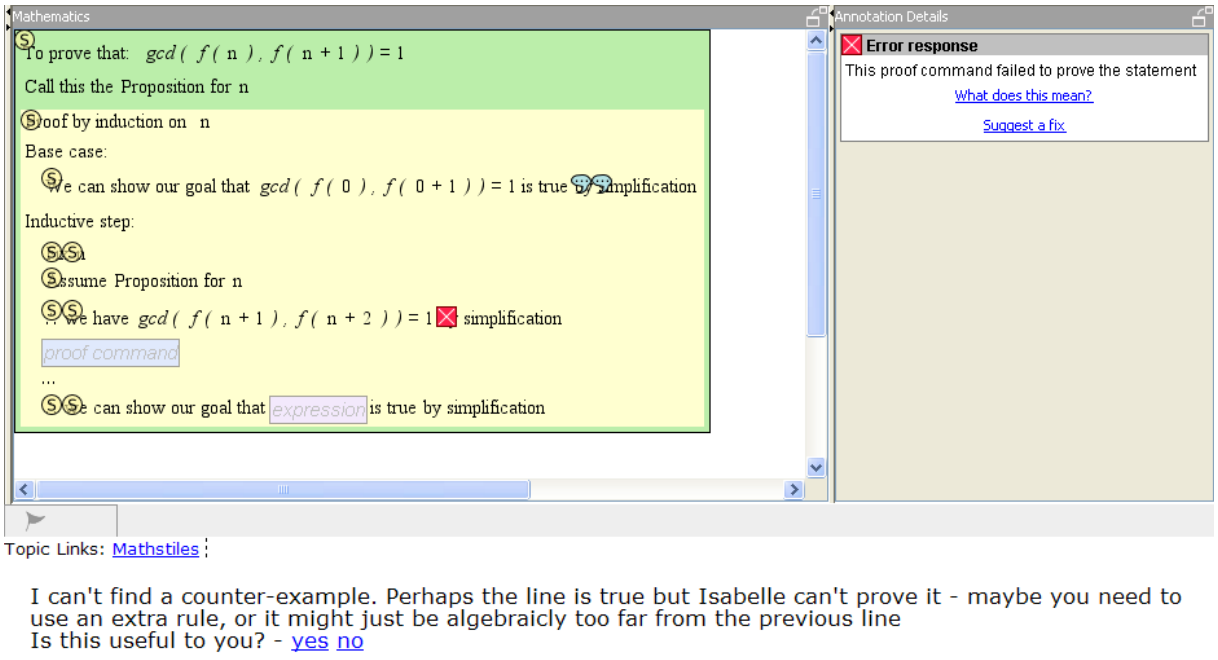
Error response
This proof command failed to prove the statement
What does this mean?
Suggest a fix

Topic Links: Mathstiles

This counter-example shows the line is wrong:
n=0
Is this useful to you? - yes no

Figure 8.5: If we insert an incorrect statement into the inductive step, the Teaching Script can help identify a counter-example.

Of course the error "*This proof command failed to prove the statement*" can also occur if we make a true statement that we cannot prove by simplification. For example, let us try to

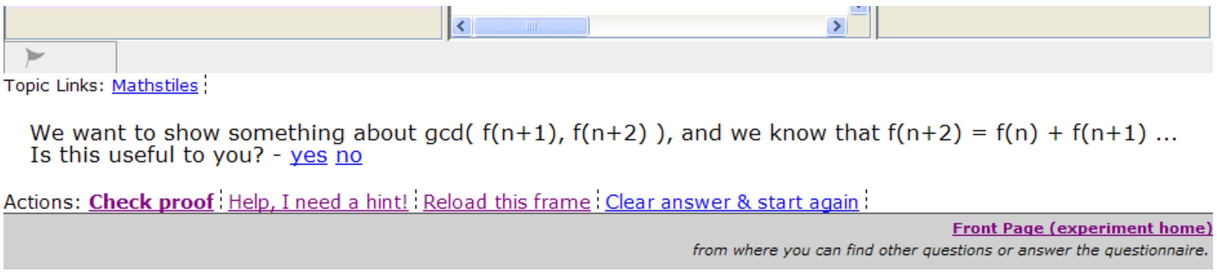

Mathematics
To prove that: gcd (f (n), f (n + 1)) = 1
Call this the Proposition for n
Proof by induction on n
Base case:
We can show our goal that gcd (f (0), f (0 + 1)) = 1 is true by simplification
Inductive step:
Assume Proposition for n
∴ we have gcd (f (n + 1), f (n + 2)) = 1 simplification
proof command
...
We can show our goal that expression is true by simplification
Annotation Details
Error response
This proof command failed to prove the statement
What does this mean?
Suggest a fix
Topic Links: Mathstiles

I can't find a counter-example. Perhaps the line is true but Isabelle can't prove it - maybe you need to use an extra rule, or it might just be algebraicly too far from the previous line
Is this useful to you? - yes no

Figure 8.6: The feedback given when no counter-example can be found. It can be difficult to ascertain why Isabelle/HOL failed to prove a statement, so the feedback tries to encourage the student to take smaller steps. (There is no straightforward definition of what "algebraically far" means – this message is simply a way of encouraging students to make each line of the proof resemble the previous line more closely. The kinds of reasoning steps that the proof can make, however, are discussed in Section 8.4.)

immediately prove that $GCD(f(n+1), f(n+2)) = 1$. This is certainly true – in fact it is almost exactly the goal for the inductive step – but it cannot be proved automatically using the simplifier. The message returned from the helper function is shown in Figure 8.6.

The tile tray has been hidden in Figures 8.3 to 8.6 in order to fit the screenshots on the page. Referring back to Figure 8.1, however, we can see that we are given the rules $GCD(m, m+n) = GCD(m, n)$ and $GCD(x, y) = GCD(y, x)$. Also, if we are stuck at this point and click the "*Help, I need a hint*" link, we receive a useful message, shown in Figure 8.7.



We want to show something about gcd(f(n+1), f(n+2)), and we know that f(n+2) = f(n) + f(n+1) ...
Is this useful to you? - yes no

Actions: Check proof | Help, I need a hint! | Reload this frame | Clear answer & start again |

Front Page (experiment home)
from where you can find other questions or answer the questionnaire.

Figure 8.7: The teaching script makes a suggestion if we click *Help, I need a hint*.

This suggests that we should substitute $f(n) + f(n+1)$ for $f(n+2)$ in our goal and see if any of the rules we are given can help us. The proof from here continues:

$\therefore$ we have $GCD(f(n+1), f(n+2)) = GCD(f(n+1), f(n+1)+f(n))$ by simplification

$\therefore$ with $GCD(m, m+n) = GCD(m, n)$
we have $GCD(f(n+1), f(n+2)) = GCD(f(n+1), f(n))$ by simplification

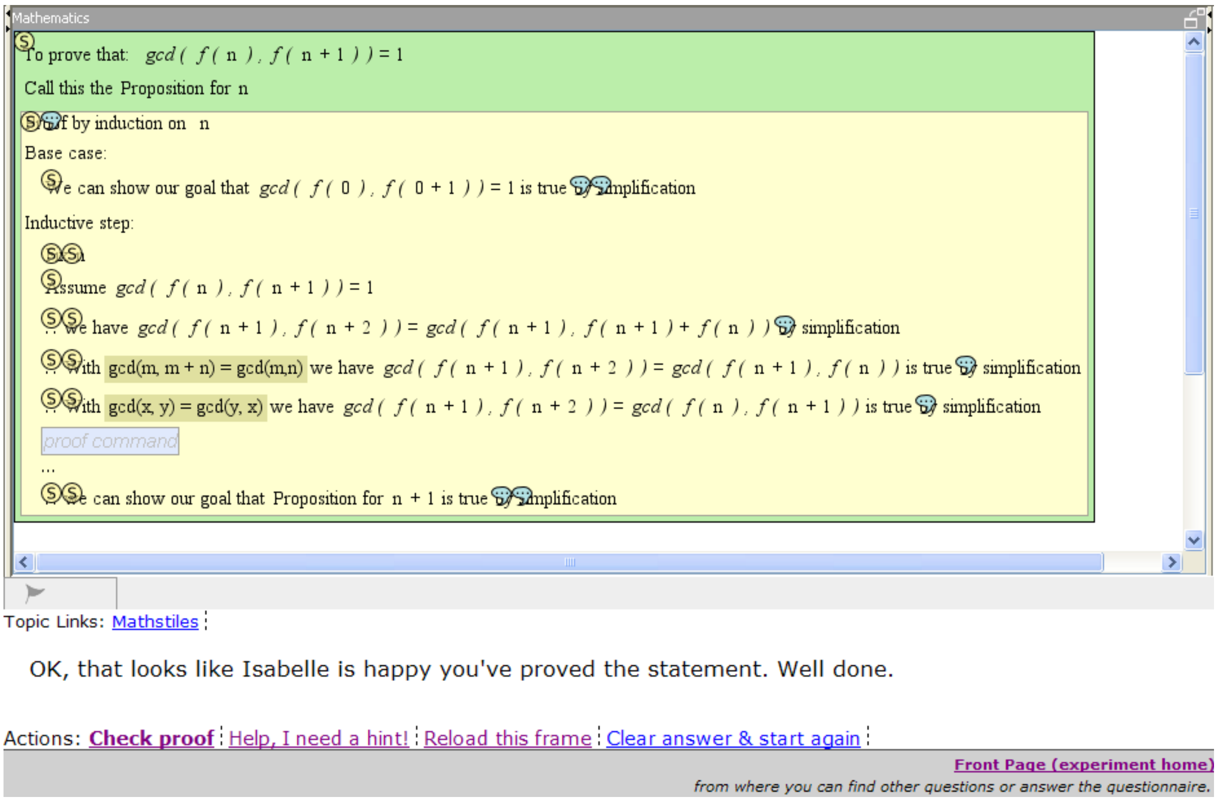
$\therefore$ with $GCD(x, y) = GCD(y, x)$
we have $GCD(f(n+1), f(n+2)) = GCD(f(n)), f(n+1))$ by simplification

This is a kind of backward proof. We wish to show that $GCD(f(n+1), f(n+2)) = 1$, so we have taken the left hand side of that equality and, by applying various rules, we have shown that it equals the left hand side of the equality from the step assumption: $GCD(f(n), f(n+1))$. In the step assumption, we assumed that $GCD(f(n), f(n+1)) = 1$, so therefore we can also conclude that $GCD(f(n+2), f(n+2)) = 1$

However, we are still not quite at our goal. Just as our goal statement for the base case involved substituting 0 for n in the proposition, so our goal in the step case involves substituting $n+1$ for n. Our actual goal line then appears as:

$\therefore$ we can show our goal that $GCD(f(n+1), f(n+1+1)) = 1$ by simplification.

Alternatively, we can use the shortcut "*Proposition for* $n+1$". This is shown in Figure 8.8. This figure also shows that the teaching script has registered the annotation from Isabelle/HOL stating that the theorem has been proved, and a congratulatory message is displayed.


Mathematics

To prove that: $gcd(\ f(\ n\ ),\ f(\ n+1\ )\ ) = 1$
Call this the Proposition for n
Proof by induction on n
Base case:
We can show our goal that $gcd(\ f(\ 0\ ),\ f(\ 0+1\ )\ ) = 1$ is true by simplification
Inductive step:
Fix n
Assume $gcd(\ f(\ n\ ),\ f(\ n+1\ )\ ) = 1$
∴ we have $gcd(\ f(\ n+1\ ),\ f(\ n+2\ )\ ) = gcd(\ f(\ n+1\ ),\ f(\ n+1\ ) + f(\ n\ )\ )$ by simplification
∴ with gcd(m, m + n) = gcd(m,n) we have $gcd(\ f(\ n+1\ ),\ f(\ n+2\ )\ ) = gcd(\ f(\ n+1\ ),\ f(\ n\ )\ )$ is true by simplification
∴ with gcd(x, y) = gcd(y, x) we have $gcd(\ f(\ n+1\ ),\ f(\ n+2\ )\ ) = gcd(\ f(\ n\ ),\ f(\ n+1\ )\ )$ is true by simplification
proof command
...
∴ we can show our goal that Proposition for n + 1 is true by simplification

Topic Links: Mathstiles

OK, that looks like Isabelle is happy you've proved the statement. Well done.

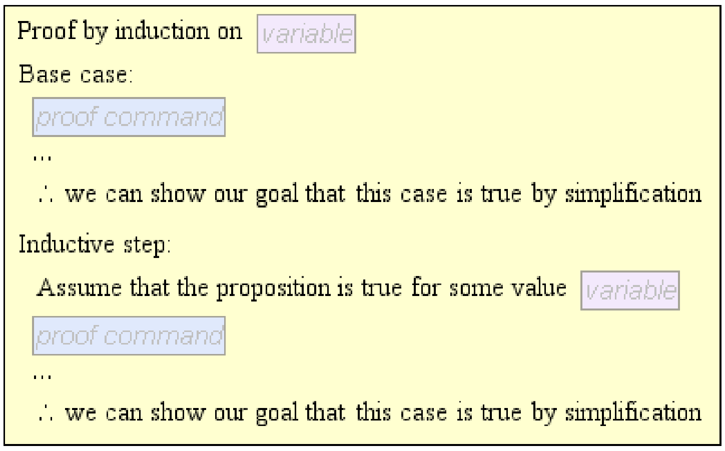
Actions: **Check proof** | Help, I need a hint! | Reload this frame | Clear answer & start again

Front Page (experiment home)
from where you can find other questions or answer the questionnaire.

Figure 8.8: The completed proof.

8.2 Proof tiles

One use for definable tiles is to expose to students what they need to do to fully answer an exam question – for example, what is needed to complete an induction proof, or how to show that a set relation is an equivalence relation. Tiles can be defined that include sockets for each section the student is expected to include to complete the induction proof or show the equivalence relation. A tile for natural induction is shown in Figure 8.9, along with its Isar translation. It is implemented as a nest of tiles, but some of them are marked as unselectable (and so cannot be taken out of the parent tile), so to the user it appears to be a single tile. The tile contains a socket for the student to fill in the induction variable. Beneath that is a section for the base case. This contains an expanding socket list for the proof steps the student will take to show the base case. The final goal step has already been filled in for this particular tile using the shortcut "*this case*" as the goal statement. The reason why this shortcut is sometimes used is described in Section 8.6. A second section in the tile is provided for the inductive step case.



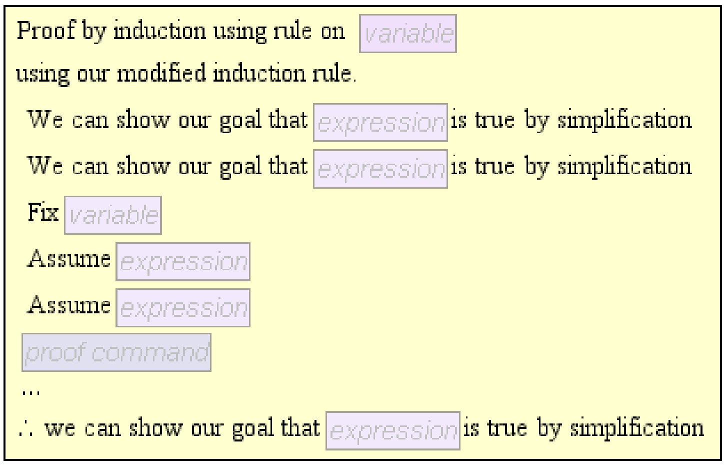
```
Proof by induction on variable
Base case:
  proof command
  ...
  ∴ we can show our goal that this case is true by simplification
Inductive step:
  Assume that the proposition is true for some value variable
  proof command
  ...
  ∴ we can show our goal that this case is true by simplification
```

proof (induct *variable* rule: altInduct)
case base
proof commands
with prems show ?case by simp
next
case (step *variable*)
proof commands
with prems show ?case by simp
qed

Figure 8.9: A tile for natural induction that is used in Section 8.7, and its Isar translation.

In the Isar code of Figure 8.9, notice the text "`rule:  altInduct`". This is not represented anywhere on the tile. This is a small example of how question-specific code can be hidden in the Isar conversion of tiles. In this case the reason for the alteration is simply that Isabelle/HOL's default induction rules use the successor function and consider cases 0 and $Suc(n)$, whereas for this question I wanted students to reason with cases 0 and $n + 1$. I therefore hid an alternative induction rule in the conversion script for the question, and set the induction tile to use it.

The induction tile in Figure 8.9 is not intended to be the only induction tile in the system. For example, Figure 8.10 shows an induction tile that is used in some questions about the Fibonacci sequence. For this tile, the induction scheme uses the definition of the Fibonacci sequence. So,


Proof by induction using rule on variable
using our modified induction rule.
We can show our goal that expression is true by simplification
We can show our goal that expression is true by simplification
Fix variable
Assume expression
Assume expression
proof command
...
∴ we can show our goal that expression is true by simplification

```
proof (induct variable rule:  fibInduct)
show expression by simp
next
show expression by simp
next
fix variable
assume expression
assume expression
proof commands
with prems show expression by simp
qed
```

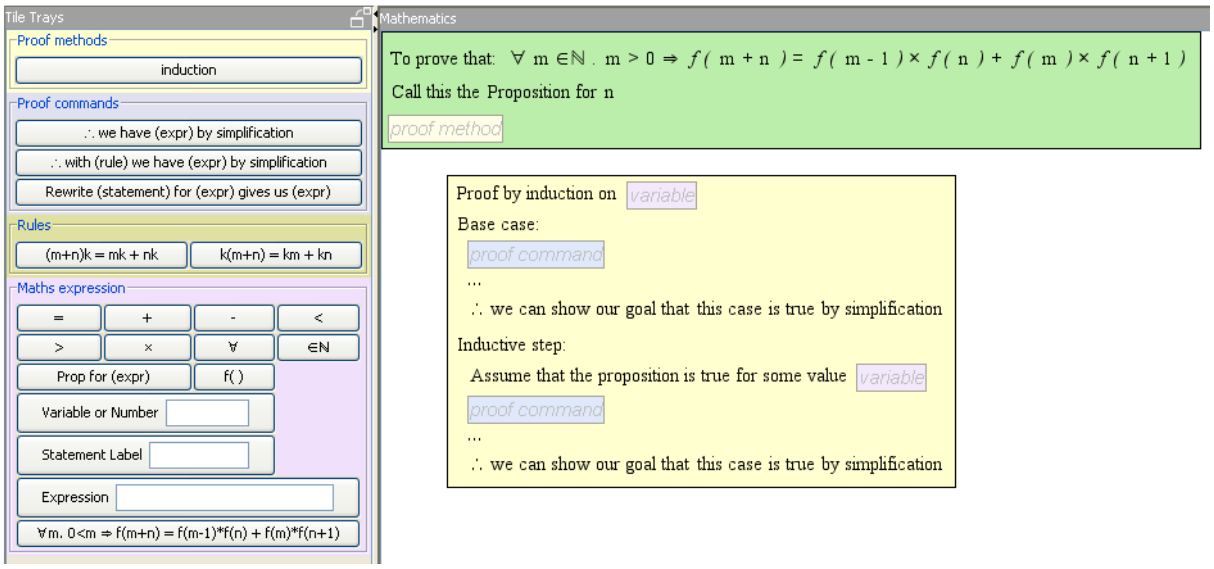
Figure 8.10: A tile for induction over the Fibonacci sequence, and its Isar translation.

there are base cases for $f(0)$ and $f(1)$ and the inductive step must make assumptions for $f(n-1)$ and $f(n-2)$. The tile also uses the induction proof method slightly differently in Isar. In Figure 8.9, the tile used the Isar case labels "`case base`" and "`case (step` *`variable`*`)`"; these cause Isabelle/HOL to make the appropriate assumptions at the inductive step automatically. In Figure 8.10, however, the student is asked to fill in the assumptions explicitly, and they are translated into `fix` and `assume` commands. The tile in Figure 8.10 also asks the student to write the goal statement and does not use the "*this case*" short cut. The straightforward example in Section 8.1, meanwhile, used an induction tile over the Natural numbers that similarly asked students to fill in the step assumption and the goal statement.

It is important to note, however, that socketed tiles are not *proof sketches* in the way that the automated reasoning community uses the term. Proof sketches [Lam95, Wie04] are proofs with some of the low level reasoning omitted to make the essence of the proof more readable. The main reasoning steps are shown in full in a proof sketch. Proof tiles, meanwhile, are syntax templates that do not contain any of the statements in the proof until the student fills them in.

8.3 Colour Coding

Although MathsTiles does not support a formal type system, it can provide the user with a few hints. In the proof exercises, I colour coded the sockets of tiles, and colour coded the background of sections of the tile tray to match. This is illustrated in Figure 8.11.



To prove that: ∀ m ∈ℕ . m > 0 ⇒ *f* (m + n) = *f* (m - 1) × *f* (n) + *f* (m) × *f* (n + 1)

Call this the Proposition for n

proof method

Proof by induction on variable

Base case:

proof command

...

∴ we can show our goal that this case is true by simplification

Inductive step:

Assume that the proposition is true for some value variable

proof command

...

∴ we can show our goal that this case is true by simplification

Figure 8.11: A tile containing a coloured socket with background text, indicating what kind of tile should be dropped into it. The buttons with the same background colour produce the right kind of tile for the socket

There are four different socket colours used. The pink sockets are for expressions. These correspond to the inner HOL syntax in Isabelle, whereas the other three colours all correspond to aspects of the outer Isar syntax. As Isabelle/HOL works through the proof, its Isar Virtual Machine [Wen05] moves between two modes that describe what kind of operation is expected next. In the *proof(state)* mode, the proof is expected to state new assumptions, goals, and intermediate results. The blue sockets and buttons are "proof commands" that correspond to this mode. In the *proof(prove)* mode, the proof is expected to justify a goal or result that it has just stated. The yellow sockets and buttons are "proof methods" that correspond to this mode. (The Isar VM has a third mode, *proof(chain)*, that the proof exercises do not use.) The khaki sockets and buttons are for statement labels and rule names.

The colours were picked arbitrarily. The decision to colour code these four categories, however, came from informal observations when volunteers first tried the proof system, before the evaluation trials. I noticed that users would often insert an expression as a line of proof, in either the base case or the step case, without enclosing it in a proof command such as "∴ *we have ... by simplification*". This happened even if they had written several previous lines correctly, and suggested that it was not noticeable enough that a proof command was needed. Although it did not happen in the pre-trials, there was also the danger that students would think they could refer to a rule by building its expression rather than selecting a rule label from the *Rules* part of the tile tray – for example, constructing $(m + n) \times k = m \times k + n \times k$ from tiles rather than selecting the "`(m+n)k = mk + nk`" rule label in Figure 8.11. Finally, I decided it was important to make the distinction between proof commands (making new statements of truth) and proof methods (justifying those statements) clear.

Dark green, meanwhile, has been used as a colour code for the question tile – the unselectable and indelible tile that describes the statement to be proved and contains an empty socket waiting for the proof.

8.4 Reasoning Step Size

Answering a proof exercise is a very different situation from professional or research use of a theorem prover. In professional use, the user should be able to use advanced automated proof-finding techniques to make his or her work easier. In a proof exercise, however, the automated proof-finding techniques the student can use must be limited because the student is supposed to answer the question, not the prover. The prover should only be able to take "obvious steps".

The approach I have taken is to limit the student to only using Isabelle/HOL's simplifier, through the Isar `simp` method ("*by simplification*" in the MathsTiles proofs). The simplifier can handle many simple steps, such as algebraic rearrangements, but cannot automatically solve the proof exercises from the Discrete Mathematics course.

The simplifier repeatedly applies a set of rewrite rules (called the *simpset*) to the current goal statement. A rewrite rule describes a pattern that might match part of the goal statement, and states what it should be transformed to. Each rewrite rule is known to be formally correct when it is applied. It might be an assumption or a lemma that has already been shown to be true in the current proof, or it might be a theorem from one of Isabelle/HOL's libraries, or it might come from the definition of a function. For example, the definition of the Fibonacci sequence in Section 8.1 produces a number of rewrite rules, including that $f(0)$ can be rewritten as 0. Additionally, Isabelle/HOL's simplifier can call upon a number of built-in methods for handling arithmetic expressions.

Allowing only `simp` also provides a "configurable notion of triviality" because rules can be added or removed from the simplifier – effectively configuring which rules are considered trivial. This can be used to force the student to be explicit about steps that are considered important for a particular question.

8.5 Annotations

As described in Chapter 5, when proofs are executed in Isabelle/HOL, the responses are collected as annotations. Figure 8.12 shows a matcher from one of the Conversion Scripts. The `output.append(...)` calls append PGIP-formatted [ALW05] Isar commands to the Broker's buffer. The `processor.talk(...)` calls then tell the Broker to write its buffer out to Isabelle/HOL and collect the responses as annotations. The annotations are associated with the tile that is passed into `processor.talk(...)`. Usually, this is "`it`", which is the tile the matcher is processing. So, choosing which matchers should call `processor.talk(it)` selects where the annotations will appear.

The annotations are shown first as small icons on the tiles. These annotations are the reason why the induction tile in Figure 8.9 is implemented as an inseparable nest of tiles: although the nest behaves to the user like a single tile, the annotations need to be marked against the commands that caused them. For example, the proof state in the base case is different from the proof state in the inductive step. The annotation types are:

- Ⓢ Proof state – these annotations let the user see what goals need to be proved at this stage of the proof, and what premises are being used.
- 💬 Comment – non-error comments, such as saying that a goal has been successfully shown.
- ☒ Error – faults Isabelle/HOL has found with the proof, or errors in syntax.

```
processor.matcher(MATHSTILES_NAMESPACE, "tile",
{it.getAttribute("definition")=="proofs:inductionNatManual"},
{
  output.append("<proofstep>proof (induct ");
  processor.process(it, "mt:socket[@name='variable']);
  output.append(" rule: altInduct)</proofstep>");

  processor.talk();

  processor.process(it, "mt:socketList[@name='step list']");
  processor.process(it, "mt:socket[@name='show']");

  output.append("<proofstep>qed</proofstep>");

  processor.talk();
});
```

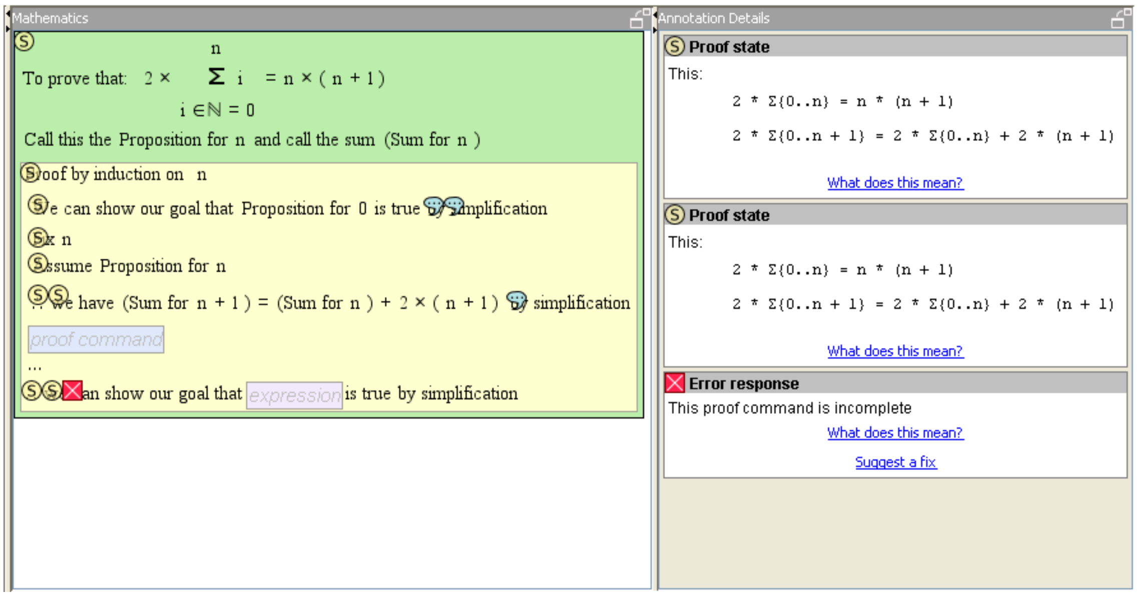
Figure 8.12: A "matcher" (pattern + template) for one kind of induction tile. The second (large) code closure describes the procedural actions to take for these tiles. `it` refers to the document element that has been matched: the tile. The base case and step assumption are implemented as unselectable tiles contained within the the `step list`. Consequently, their Isar code is not produced by this matcher but by their own separate matchers.

Clicking on the icons gives more detail of the annotation in a separate pane, as shown in Figure 8.13.

The responses from the prover are post-processed in the Broker in order to make the messages more understandable to the student. They are also assigned topic keys, which refer to the content model described in Chapter 4. The "*What does this mean?*" link in the annotation pane looks up a the associated topic in the book. Error annotations have a "*Suggest a fix*" link underneath them. Clicking this link calls an advice function in the Teaching Script for the error's topic.

The Teaching Script superclass for proof questions contains some advice functions for common errors topics. For example, it includes a helper function for the "*Proof command failed*" error message that will try a number of different values for variables to try to find a counter-example that would show the proof line was untrue rather than just unproven. This finds the relevant state annotation that contains the premises and goals of the failed command and parses each goal and premise. It attempts to find numbers which match the premises but do not match the goal statement. An advantage it has over just using Isabelle/HOL's in-built mechanism for finding counter examples is that the Teaching Script can use a different definition of a function. For example, using the equation for the nth Fibonacci number instead of the recursive definition of the sequence.

As described in Section 5.1.3, the advice function to call is chosen by an algorithm in the Teaching Script. This collects all the registered advice functions for the topic – these may come from a Teaching Script superclass or from the Teaching Script for this particular question. It then checks which advice functions are relevant, according to their relevance functions, and then selects a function to call based on whether previous students found it useful.



To prove that: $2 \times \sum_{i \in \mathbb{N} = 0}^{n} i = n \times (n + 1)$

Call this the Proposition for n and call the sum (Sum for n)

roof by induction on n

e can show our goal that Proposition for 0 is true mplification

x n

ssume Proposition for n

∴ we have (Sum for n + 1) = (Sum for n) + 2 × (n + 1) simplification

proof command

...

an show our goal that expression is true by simplification

Proof state

This:

```
2 * Σ{0..n} = n * (n + 1)
2 * Σ{0..n + 1} = 2 * Σ{0..n} + 2 * (n + 1)
```

What does this mean?

Proof state

This:

```
2 * Σ{0..n} = n * (n + 1)
2 * Σ{0..n + 1} = 2 * Σ{0..n} + 2 * (n + 1)
```

What does this mean?

Error response

This proof command is incomplete

What does this mean?

Suggest a fix

Figure 8.13: The responses from Isabelle/HOL are marked on the proof tiles as annotation icons; these annotations can then be shown in full in the annotation pane by clicking on their icons. The annotations disappear when tiles are dropped into or pulled out of a socket. (Since the user has already placed the tiles, and so knows what they are, the fact that the icons can obscure some of the text on the tile is less of a problem than it might appear from the picture.)

8.6 Two Design Compromises

In Chapter 6, I described design goals that the student should have to write the statements in a proof, and that the proof should resemble what students write on paper. In this section, I describe two design compromises I made in this area.

8.6.1 The student does not always have to write the goal statements

Referring back to the induction tile in Figure 8.9, the goal statements for the base case and inductive step are simply the shortcut "*this case*". The student has not been forced to write them.

The reason why this shortcut is sometimes used is that when we tell Isabelle/HOL that we are using induction or proof by cases, Isabelle/HOL automatically works out what the goals need to be for each of the cases. Students, if they were allowed to write in the goal, might write it in a way that a human would consider equivalent but that is very slightly different to the goal Isabelle/HOL calculated – for example swapping the sides of an addition. This would then cause the goal statement to fail. Isabelle/HOL expects the goal statement to be shown exactly as calculated, and will not allow something to be shown that is a few steps of logic away instead.

A possible workaround for this would be for the tile not to use the `show` command for the user's goal, but to treat it as just another `have` command and then hide a command to show the real goal by simplification in the Conversion Script. This would allow the user to put in a goal that was "trivially close" to the goal and the proof would succeed. Unfortunately, for goals that Isabelle/HOL's simplifier can prove from the definition, such as $\sum 0..0 = 0$, this would also allow the user to write in a true but irrelevant statement, such as $1 = 1$, as the goal and the hidden proof command would still prove the real goal. The human notion of a "trivial step" is different from the notion of whether a statement is equivalent to the goal.

Section 9.4 discusses some potential long-term solutions to this issue. For the usability trial, described in the next chapter, however, students were given two tiles that would make stating the goal more straightforward. In most questions, a "*Proposition for ...* " tile was available, as used in the example in Section 8.1. This tile provides a pattern for writing the proposition from the question, with a particular value or expression inserted. For example, the goal of the base case of an induction over the naturals would be the "*Proposition for* 0", and in the inductive step we would assume the "*Proposition for* n" is true and attempt to show the "*Proposition for* $n + 1$" must also be true. In one question, however, the induction tile had a "*this case*" tile fixed in its goal sockets, so in that question students did not have to write the goal statement at all.

8.6.2 The proof is checked linearly.

The student is free to write the proof in any order using MathsTiles. However, because the proof is translated into Isar, an error in the proof is likely to cause every following line of proof to fail. These follow-on errors could be an unhelpful distraction from the original (causative) error, so when the proof is checked, the Broker stops collecting annotations after the first error. This means that the student gets no feedback on correctness for the lines after the first error. While the interface does not prevent the student from constructing the proof in any order, the system provides much stronger support for starting at the beginning of the proof and working towards the end.

8.7 A Difficult Example

This example is part of a question from the 2004 written exam sat by first year undergraduates in the Computer Science tripos. It is a proof exercise that is technically more difficult in Isar. It is described here to show how a question author, by adjusting the question and the proof script, can set a question up so that students will not encounter some of the technicalities.

The student is again given a definition of the Fibonacci sequence (the same definition as is given in Section 8.1), and is asked to prove by induction that $f(m + n) = f(m - 1) \times f(n) + f(m) \times f(n + 1)$ for all $m > 0$, where $f(n)$ corresponds to the nth element of the Fibonacci sequence. A rough paper proof that resembles the MathsTiles notation is shown in Figure 8.14. The completed MathsTiles version of the proof is then shown in Figure 8.15. However, there is a difference between the paper proof and the MathsTiles proof: in the paper proof both n and m are explicitly universally quantified; in the MathsTiles proof m is explicitly universally quantified, but n is not – although it is *implicitly* universally quantified.

Practically, the reason for the difference is that as a question author I initially wrote the proof in Isar with both variables explicitly universally quantified, and the proof failed. Removing the quantifier from n allowed the proof to succeed, but if I removed the quantifier from m as well, the proof failed again. In each case, I decided the reason for the failure was too technical to expose to first year undergraduate students. So, by writing the question with m explicitly universally quantified and n not, I forced the students answering the question to take the path that succeeds.

The reason why n must not be universally quantified is that in Isabelle's HOL logic, induction is only permitted over *free variables* [NPW05]. A free variable acts as a place marker that

To prove that $\forall m, n \, . \, m > 0 \longrightarrow f(m+n) = f(m-1) \times f(n) + f(m) \times f(n+1)$

Proof by induction on n

Base case:

We can show $\forall m \, . \, f(m+0) = f(m-1) \times f(0) + f(m) \times f(0+1)$ by simplification

Inductive step:

Fix n

Assume `A`: $\forall m \, . \, m > 0 \longrightarrow f(m+n) = f(m-1) \times f(n) + f(m) \times f(n+1)$

With `A`, substituting $m+1$ for m,
we have $f(m+1+n) = f(m) \times f(n) + f(m+1) \times f(n+1)$ by simplification

$\therefore$ With $m > 0 \longrightarrow f(m+1) = f(m) + f(m-1)$
we have $f(m+1+n) = f(m) \times f(n) + (f(m) + f(m-1)) \times f(n+1)$ by simplification

$\therefore$ With $(m+n)k = mk + nk$
we have $f(m+1+n) = f(m) \times f(n) + f(m) \times f(n+1) + f(m-1) \times f(n+1)$ by simplification

$\therefore$ With $km + kn = k(m+n)$
we have $f(m+1+n) = f(m) \times (f(n) + f(n+1)) + f(m-1) \times f(n+1)$ by simplification

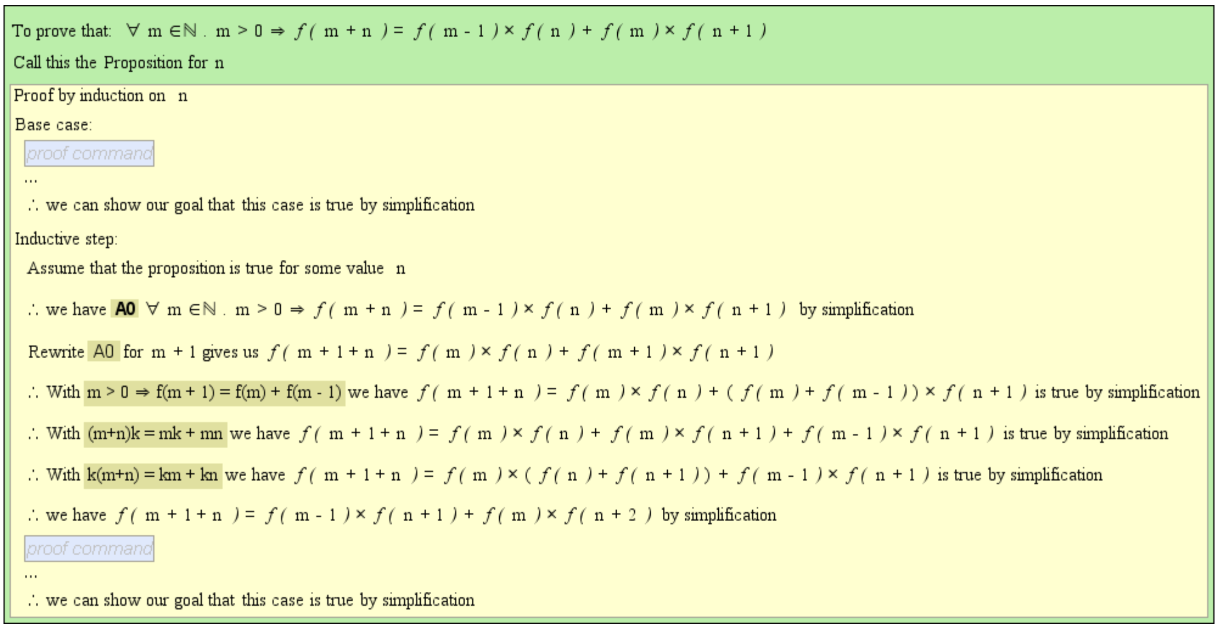
$\therefore$ With $m > 0 \longrightarrow f(m+1) = f(m) + f(m-1)$
we have $f(m+1+n) = f(m) \times f(n+2) + f(m-1) \times f(n+1)$ by simplification

$\therefore$ We have
$\forall m \, . \, m > 0 \longrightarrow f(m+1+n) = f(m-1) \times f(n+1) + f(m) \times f(n+1+1)$
by simplification

$\therefore$ Our final goal, that $\forall m \, . \, m > 0 \longrightarrow f(m+(n+1)) = f(m-1) \times f(n+1) + f(m) \times f((n+1)+1)$ can be shown by simplification.

Figure 8.14: A proof of the question that makes sense on paper. In the inductive step, we perform a forward proof: we take the step assumption and use it deduce further statements until we can finally conclude that the goal statement must also be true. (Again, "*by simplification*" asserts that the statement can be shown automatically using the set of term rewriting rules and arithmetic procedures that are available to Isabelle/HOL's simplifier – see Section 8.4.) The proof fails in Isabelle/HOL because in Isabelle's HOL logic, induction is only permitted over free variables, whereas in this paper proof, n is bound by a universal quantifier.

can be substituted with any other expression later, subject only to type-checking (eg, a Boolean cannot be substituted for a Natural number). In the example question, we wished to prove a proposition, let us call it $P(m, n)$, is true for all $m > 0$ and for all $n \in \mathbb{N}$. To prove this by induction in Isabelle/HOL, we should in fact prove that $P(m, x)$ is true for all $m > 0$ and the free variable x. This gives us a free variable to induct over. We should then add a final generalisation step, in which we say "since $P(m, x)$ is true for $m > 0$ and the free variable x, we can substitute the universally quantified variable n for x, and so $P(m, n)$ is true for all $m > 0$ and all $n \in \mathbb{N}$".



To prove that: $\forall$ m $\in \mathbb{N}$. m > 0 ⇒ *f* (m + n) = *f* (m - 1) × *f* (n) + *f* (m) × *f* (n + 1)

Call this the Proposition for n

Proof by induction on n

Base case:

proof command

...

∴ we can show our goal that this case is true by simplification

Inductive step:

Assume that the proposition is true for some value n

∴ we have **A0** $\forall$ m $\in \mathbb{N}$. m > 0 ⇒ *f* (m + n) = *f* (m - 1) × *f* (n) + *f* (m) × *f* (n + 1) by simplification

Rewrite A0 for m + 1 gives us *f* (m + 1 + n) = *f* (m) × *f* (n) + *f* (m + 1) × *f* (n + 1)

∴ With m > 0 ⇒ f(m + 1) = f(m) + f(m - 1) we have *f* (m + 1 + n) = *f* (m) × *f* (n) + (*f* (m) + *f* (m - 1)) × *f* (n + 1) is true by simplification

∴ With (m+n)k = mk + mn we have *f* (m + 1 + n) = *f* (m) × *f* (n) + *f* (m) × *f* (n + 1) + *f* (m - 1) × *f* (n + 1) is true by simplification

∴ With k(m+n) = km + kn we have *f* (m + 1 + n) = *f* (m) × (*f* (n) + *f* (n + 1)) + *f* (m - 1) × *f* (n + 1) is true by simplification

∴ we have *f* (m + 1 + n) = *f* (m - 1) × *f* (n + 1) + *f* (m) × *f* (n + 2) by simplification

proof command

...

∴ we can show our goal that this case is true by simplification

Figure 8.15: The solution to the difficult question, in MathsTiles.

However this would be a very difficult and subtle concept to explain to a novice student, as it is a difference between the way the students' first year Discrete Mathematics lecture notes [Rob06] describe induction and the way the proof assistant handles induction. The lecture notes describe mathematical induction as a method to prove a proposition true *"for every natural number"*, rather than for "*a free variable of type Natural*".

The reason why m must be explicitly universally quantified in the proposition is so that we can legitimately substitute $m + 1$ for m in the inductive step assumption. At the beginning of the inductive step, we have a statement $A(m, n)$ that we are going to assume. If we assume $\forall m \in \mathbb{N}.A(m, n)$ is true then we can legitimately deduce that $\forall m \in \mathbb{N}.A(m+1, n)$ is also true. However, if we simply assume that $A(m, n)$ is true, and do not universally quantify m, it is not valid to conclude that $A(m + 1, n)$ is also true.

There is a subtlety that would be harder to explain to first year students, however. Free variables are implicitly universally quantified, in that they can stand for any expression of the same type. For example, when I described why n had to be a free variable, I explained that we could add a final generalisation step to introduce the quantifier – replacing a free variable with a universally quantified variable. And yet that implicit universal quantification does not allow us to say $A(m, n) \longrightarrow A(m + 1, n)$ if m is a free variable. We also cannot insert a generalisation step to universally quantify m within the inductive step. If m is a free variable in the proposition, then we can only add a generalisation step to quantify m after we have proved the proposition, and not in the middle of its inductive proof.

CHAPTER 9

Evaluation

In Sections 9.1 to 9.6 of this chapter, I describe a qualitative evaluation of the MathsTiles/Isar number theory proof exercises. The evaluation shows some of the advantages of the system but also presents a number of challenges that remain to be solved, and helps to uncover "why students find maths hard".

Additionally, in Section 9.7, I describe in more detail the differences between the Intelligent Book and ActiveMath. ActiveMath is the research project that is most similar in aims to the Intelligent Book, and so it is worth assessing how the system I have developed differs from it.

9.1 Overview

My goal in evaluating the system was twofold. By asking students and others who have no experience of automated proof to attempt the exercises, I wish to see whether novices can make progress with the exercises with a bare minimum of training. More importantly, I wish to understand the usability issues that arise from the system, and whether they are insurmountable and a different approach is required, or whether they suggest fruitful avenues of further inquiry.

To this end, with the assistance of undergraduate intern Sparsh Gupta, I performed a user trial and qualitative usability study using the Cognitive Dimensions of Notations (CDs) framework [GP96, BG03]. CDs provide a formalised vocabulary for discussing usability issues, with sixteen "dimensions" that can affect usability. An example of a Cognitive Dimension is *"viscosity"*, which is the question of how difficult is it to make changes to previous work using the interface. The CDs Framework provides means for considering *secondary notations, helper devices*, and *redefinition devices*, but in this study we only examined the primary notation: the MathsTiles proofs.

I chose CDs for the evaluation because it is a technique I am familiar with, and I was confident that it could meet my objectives. However, I also believe that any suitable evaluation mechanism would probably have produced similar results to the ones I present here.

Two methods were used to collect usability data:

1. A server containing an introduction to the system and six proof exercises was made pub-

lically available, and its use over three weeks in July 2006 was examined. A range of users were asked to try the system, including Cambridge undergraduate students, undergraduate students from other universities, postgraduate tutors of Discrete Mathematics, and other interested parties. The examined exercises were about the Greatest Common Denominator $GCD(a, b)$ and the Fibonacci series $f(n)$. The proofs exercises were:

(a) Prove that $2 \times \sum 0..n = n \times (n + 1)$, by induction on the Natural numbers. This was the introductory example for which a walkthrough was given.

(b) Prove that $GCD(f(n), f(n + 1)) = 1$, by induction on the Natural numbers.

(c) Prove that $n > 0 \Longrightarrow GCD(n \times k + m, n) = GCD(m, n)$, by assuming the left hand side of the *implies* is true and showing the right hand side must follow.

(d) Prove that $f(n+k+1) = f(k+1) \times f(n+1) + f(k) \times f(n)$, by induction using a different induction rule. There are two base cases: for 0 and 1. In the inductive step, the student should assume the proposition is true for some n and $n + 1$ and prove that it must also be true for $n + 2$.

(e) Prove that $\forall m.m > 0 \Longrightarrow f(m + n) = f(m - 1) \times f(n) + f(m) \times f(n + 1)$, by induction on the Natural numbers.

(f) Prove that $GCD(f(n + m), f(m)) = GCD(f(n), f(m))$, by considering the possible cases for m (either $m = 0$ or $m = p + 1$ for some $p \in \mathbb{N}$).

Three kinds of training items were provided. Two Flash videos, totalling just over three minutes in length, showed how to use the MathsTiles interface. An "introductory chapter" to the exercises, three pages long, explained similar material to the videos (for participants who might not have had the Flash plugin installed). A walkthrough described how to solve the first and simplest question, with screenshots.

The comments, feedback, and requests for help from users were coded against the CDs Framework by two researchers.

2. To identify issues that novice participants might be prone to miss or unable to articulate, the system was assessed against a Cognitive Dimensions of Notations questionnaire [BG07]. This was carried out both by myself and by the undergraduate intern, who had worked with the system for two months. The collected comments were also passed to an expert in CD analysis for informal review.

9.2 Numerical Results

The numerical results from the trials are shown in Table 9.1.

While very few participants indicated whether or not they were students, from examining their email addresses and how they became involved with the trials I confidently identified 44 of the participants as students. Of the five participants who completed question two, three were students. One of the participants who completed five proofs was a student; the other two were tutors of students but had no prior experience with Isabelle/HOL.

The five participants who accessed question two but were judged "not to have made a serious attempt" put fewer than six tiles on the canvas, placed all their tiles on the canvas very quickly,

Participants	Stage
83	Accessed the server and read about the system
19	Accessed the introductory question
8	Completed the introductory question
13	Accessed question two
8	Made a serious attempt at question two
5	Completed question two
3	Completed five of the six proofs

Table 9.1: The number of participants reaching each stage of the exercises.

and did not attempt to construct any expressions or place a proof method into the answer tile. From this I concluded that they played briefly with the interface, but did not attempt the proof.

On the one hand, the results are encouraging. In Chapter 6, I noted that the shortest training course in Isabelle/HOL is four sessions of ninety minutes, with 300 slides, and I was given an (unscientific) estimate that students might take ten weeks to be able to do simple things using the prover. In this trial, some novice users and students have been able to complete proof exercises despite their training being barely three minutes of videos, three explanatory Web pages, and a walkthrough of a single proof. On the other hand, however, there is a significant attrition from 83 initial participants down to three who completed five proofs, and only one of those was a student. This suggests there are still some major issues to overcome.

It is not possible, of course, to determine the reason for the attrition from 83 participants down to 8 who made a serious attempt at a proof without a walkthrough. Many of these participants may simply have been interested in looking at a new interface, but not interested in attempting a mathematical proof. On the other hand they may have been scared away by the complexity of the system. The three participants who failed question two, however, reported that they had become stuck.

9.3 Qualitative Results

In the user study, I asked participants to fill in a feedback questionnaire. However, I found that many of the participants were reluctant to fill in a questionnaire form, but were more than happy to contact either Sparsh Gupta or me informally to give us their feedback. Consequently, feeback was received by email, instant messenger, and discussions with users who came to my office or phoned me to tell me their thoughts and demonstrate the issues they were having. While this meant that feedback was received in a less controlled manner, it had the advantage of immediacy – we were able to examine the participants' question documents when the issues were reported to see the issues in practice and ensure we had not misunderstood them.

After the user feedback had been received, we conducted an analysis using the Cognitive Dimensions questionnaire.

The full table of issues identified is at the end of this chapter, in Section 9.6. Because of the informal and verbose nature of the feedback, I have rephrased many of the issues for concise presentation in the table.

For discussion purposes, I have classified these 30 qualitative statements into five categories, three of which I discuss in detail:

Non-problems.
Statements 10, 18, 21, 24, and 26.
This category lists all the positive and non-negative remarks.

MathsTiles UI (Simple).
Statements 2, 5, 6, 7, 8, 9, 11, 12, 15, 16, 17, 22, 27, and 30.
This category lists usability issues that suggest straightforward enhancements to make to the MathsTiles or Intelligent Book user interface that do not impact on the approach. For example, bug fixes are listed under this category.

MathsTiles UI (Complex).
Statements 13, 28, and 29.
This category lists usability issues with the MathsTiles and Intelligent Book interface I regard as more complex or interesting. These are discussed in Section 9.3.1.

Proof Language
Statements 4 and 14.
This category lists usability issues that specifically relate to using MathsTiles as a proof language that translates to Isar. These are discussed in Section 9.3.2.

Domain Specific (here Number Theory).
Statements 1, 3, 19, 20, 23, and 25.
This category lists usability issues I regard as inherent to the problem of freely-written student proofs in "difficult" domains such as Number Theory. These are discussed in Section 9.3.3.

The following three sections discuss the statements from the last three categories in detail. These three categories are discussed in detail because they represent complex challenges still to be overcome, whereas the first two categories do not.

9.3.1 MathsTiles UI (Complex)

Statements 13 and 28: Limitations with the expression control

In Statement 13, a user has seen that it is possible to type expressions, and has assumed that any text that appears on a tile can be typed into the expression control and recognised as a valid expression. This is an issue that to an extent has already been discussed in Section 7.7. Unfortunately, the MathsTiles applet in its current version uses a traditional generated LL(k) parser with a fixed grammar. So, it is incapable of adding the defined tiles for a question to its expression grammar.

Statement 28 describes how students wanted to insert incomplete expression fragments using the expression control. This is another case where the fixed LL(k) parsing is insufficient, but for a different reason. The design assumption had been that users would wish to type complete algebraic expressions into the box to save the effort of composing them from tiles, or would cut and paste expressions into the box from annotations. However it turns out that very often users only want to add the few tiles they need to alter an existing expression, but they still type them into the expression control. These few tiles are necessarily an incomplete expression fragment, and might or might not be parsable with the current parser. One possible solution to this would be to support placeholders in expressions (or effectively to have a syntax element for an empty socket). For example, the expression “$3 + \_$” could represent an addition where the right socket is left empty.

An interesting observation, not noted in the table, is that the users who became confused by the expression box and reported Statement 28 were trying to use the expression control in the first (tutorial) exercise. After it was reported, I removed the control from this exercise and new users did not encounter the expression box until question two. No further complaints about the expression control were received and one new user complimented it (Statement 18). This might be due to individual differences in the users, but it might suggest that when users gain even a little more experience of an interface, they become much readier to work around the limitations of newly introduced components.

Statement 29: Missing entry in the Book

In Statement 29, a user was surprised by the Intelligent Book defaulting to a Web search when it found it did not have an entry for a topic. While I ensured that participants saw an introduction to the maths problems, I did not ensure that they saw an explanation of how the book’s content features work. (There was a low-key link on the instructions page, but I deliberately did not draw attention to it). I left this particular entry out of the book curious as to whether participants would add an entry when they discovered the feature, even though they had not been explicitly taught how to. They did not.

9.3.2 Proof Language

Statement 4: Universal quantification and the Rewrite tile

In Statement 4, a student is unaware that a statement must include a universally quantified variable before it can be rewritten with a different expression substituted for that variable. This appears to reflect that either students do not yet understand the difference between a variable that has and has not been universally quantified, or they assume that all the variables in the statement are implicitly universally quantified. Unfortunately, I also found from experience of writing questions, as described in Chapter 8, that proofs run into fewer technical problems in Isabelle/HOL when the variables in the expression are not universally quantified.

Statement 14: Labelling of prior statements

Statement 14 perhaps represents a difference between the way people informally view proofs and the way formal proof languages do. The students were surprised that the prover appeared to “forget” statements that were only two lines back in the proof. When people write arguments in English, they expect the reader to remember the context of the text so far without labelling the earlier sentences they refer to. (This can be seen in the model answers to the tutorial questions shown in Chapter 10.) The “$\because$ *we have ... by simplification*” tiles that students were using in their questions, however, translate to the Isar structure “`with prems have ...  by simp`”. This uses only the previous line and the assumptions to justify the new line of proof. If any earlier lines of the proof are needed, they must be labelled and referenced explicitly.

On the one hand, this requirement to label referenced statements is an artificial form of interaction that does not match exam paper proofs. On the other hand, however, forcing students to state which previous lines they are using forces students to think about the structure of their proofs and might be considered to be educationally helpful. Perhaps a suitable approach would be to make referring to earlier statements easier by automatically labelling every proof line, and to add a visual hint to indicate that by default the proof statement only uses the immediate previous statement and the assumptions.

9.3.3 Domain Specific Issues (here Number Theory)

Statement 1: Tiles were only provided for one solution

In Statement 1, the problem is that not enough tiles have been provided to allow the student to solve the problem by a different proof strategy than the author intended. In terms of Cognitive Dimensions of Notations, there is a trade-off between *visibility* and *premature commitment* here – by providing more tiles it becomes slightly harder to identify which ones you need. In this particular case, the extra tile is a different induction tile, and providing it would be unlikely to make the tiles too hard to find. However, in cases where you need to provide extra rules to support alternate strategies, this loss of visibility could becomes a much greater problem.

The set of rules that the simplifier includes (and that the exercises consider "trivial" as described in Chapter 8) is called the *simpset*. In the second question, the simpset included some 1,570 rules. While students do not need to know what rules are in the simpset, they need to be able to ascertain what rules are *not* in the simpset. How else could they know they need to state them? The set of rules in the tile tray gives a strong visual cue as to which rules have to be stated. However, the more rules there are in the tile tray, the harder it is to spot each rule. Taking the rules off the screen (and using a query mechanism for them) does not appear to be a viable option. Students might only be able to articulate what rules are necessary for a step if the step size was very small.

Allowing students to use more complex automated methods, rather than just the simplifier, would be one possible way of resolving this issue. Referring back to Section 8.4, I restricted students to use only the simplifier because it forced them to state any "non-obvious steps" explicitly, and provided a configurable notion of triviality. However, if more complex automated methods were made available to students, there is the danger that students them to solve the question by trial and error. There appears to be an interesting trade-off between allowing students to "game the system" and making it easier for them to explore the proof.

Another possible approach might be to allow the teaching script or conversion script to infer the necessary rule. During the verification process, the script could try each rule in turn and then identify what rule was required. Only steps that involved adding a single rule would be allowed. This would have the effect that the student would need to state the steps explicitly but not the rules.

Statement 3: Proofs are fragile

Statement 3 describes how changing an early line of the proof can cause following lines to fail. Even a trivial re-ordering of additive terms in an equation can cause a rewrite rule to fail – the terms are equivalent to the student but not to the prover. There are two aspects to this. On the one hand, perhaps the system should remember which lines it has already proved, and be more reluctant to mark those lines as no longer proved. On the other hand, this could give an inaccurate proof document, where lines of proof purport to have come from one chain of reasoning, but actually come from another. Another potential solution might be to use a less rigorous theorem prover that treats "equivalence" in a manner more similar to that which the student expects. (This prover might yet need to be invented, however.)

Statements 19 and 20: Students could only take small proof steps

Statements 19 and 20 describe how the steps students can make at each line of a proof in an exercise are very small. In the proof exercises, this relates to the fact that we only allow students to use the simplifier, and we only allow them to invoke one non-trivial rule at a time. However, even if these restrictions were relaxed students still might find themselves limited in the kind of reasoning steps they can take. The reasoning steps that automated methods can make do not easily and naturally correspond to the steps that a human can make. So, just as humans can take reasoning steps that are hard to verify automatically, automated methods can also take steps that a human would find hard to follow. If we rely purely on automated reasoning to provide the model for a question, then we can only support smaller steps that both automated methods and humans can follow.

Statement 23: Students could not recognise a bug from a mistake

Statement 23 describes how a bug in the tile translation caused an error in some proofs, but students could not tell that this was due to a bug and assumed their proofs were wrong. This is perhaps an inherent problem with a teaching system in a difficult domain – because students are inexperienced with the material and the system, they find it difficult to think critically about whether the system is operating as expected. This means that very careful testing and debugging of proof questions is necessary before they are made available to students.

Statement 25: Lack of a proper progress measure

Statement 25 describes how the only visible measure of progress with a proof is the number of rule tiles that have been provided but not used yet. It would be possible to provide a more direct measurement of the student's progress by comparing it to a pre-written proof, but as with Statement 1, this raises the problem that unexpected solutions could not be supported in this way. Practically it might be appropriate for exercises to provide guidance and support for a number of pre-planned proofs, but allow unexpected proofs also to be constructed even though only limited assistance could be provided with them.

9.4 Future Work

This section describes some issues that are interesting to consider in future work that directly relate to formally verifiable proof exercises and to the usability study. More general issues that arise for future work are discussed in Chapter 11.

9.4.1 Consideration of lemmas

The evaluation exercises did not assess how students can define lemmas in their proofs. The reason why I did not consider it here is that proof exercises are often set in a number of stages. Parts (a), (b), and (c) might ask the student to prove particular useful lemmas, and then part (d) might ask the student to use those lemmas to derive an important result. In the exercises, each of these parts could be set as a separate exercise. (And indeed the final question in the study did draw together the lemmas proved in the previous questions). However, not all questions on paper are set in this broken-down style, and if a Reactive Learning Environment is to let students try out their ideas, then it is important that they should be able to take their own approaches to solving the proofs.

9.4.2 Not using a direct translation

Performing a direct translation from MathsTiles to Isar is a fairly naïve approach to the problem. It was taken on the grounds that, this being an unusual proof interface, it was important to reuse an existing and well established reasoning mechanism (so that there would not be too many novel factors impacting on the usability study). It would be perfectly reasonable instead for the Broker, when examining a line of proof, to set all of the previous statements as lemmas, define the proof line as a goal theorem, and see whether an automated tactic can prove it or not. There would need to be some careful consideration of what theorems should be given to the tactic, however, so this would move much of the problem into the configuration of the proof tool. However, it could allow the MathsTiles proofs to resemble Isar much less – there would not need to be a straightforward translation to Isar. It would also be possible to try to verify a given line of proof using more than one reasoning system.

9.4.3 Automatically set parts of a proof document

Writing proofs using tiles is currently a one-directional activity, where the student writes the proof and the system comments on it. However, where there are dependencies between elements in the proof, it may be helpful to allow the system to write or adjust parts of the proof, or to allow parts of tiles to be calculated from their surrounding rather than strictly defined in the XML. For example, if an early proof line is changed that breaks later lines of proof, perhaps the system should attempt to automatically adjust the later lines so they are no longer broken. Similarly, where an automated proof method is used to justify a statement, perhaps that automated proof method should be able to write back to the MathsTiles proof the details of the proof steps it used.

9.4.4 Configurable level of formality

It may be helpful educationally to be able to have a configurable level of formality in the prover. For example, we observed that students did not appear to understand the issues around universal quantification. What if the model could be made to temporarily forget those issues until the student was due to learn them? A common technique through school education is to teach a simplified and abbreviated version of the material first, and to introduce the complexities later.

9.5 Conclusions from the Qualitative Evaluation

The exercises appear to have enabled a few users in the study to complete formally verifiable proofs with a surprisingly small amount of training. The usability issues raised with the interface during the study do not appear to be insurmountable, although there remain a number of significant challenges these proof exercises have not addressed. For example, each of the exercises only provided the right tiles for a solution that had already been carefully checked by the teacher. This means that although students are theoretically free to “try out their ideas” in a Reactive Learning Environment, in practice they can only succeed with ideas the teacher has thought of for them.

Two participants commented informally after the study that through attempting the exercises they felt they had learnt a little more about automated proof assistants, and felt braver to try using Isabelle/HOL, where before they thought Isabelle/HOL would be too difficult to learn.

Some challenging user interface issues arise where the student’s expectation of how something should work is different from the goals of formal proof. For example, students appeared to hope that all the statements they have made so far in the proof would be remembered, and the checker would automatically determine which ones should be used to demonstrate the next statement; formal proofs, meanwhile, attempt to be explicit about their structure and which statements are involved in which steps.

Another challenge is developing automated systems that are simple enough for a student to understand roughly how they work, but that can make the same kind of steps that humans do when reasoning about a proof. The system needs to be able to verify human reasoning steps so that automated proof exercises do not have to differ too much from paper proofs. Students must be able to understand roughly how the reasoning system works because there are often proof steps that a reasoning system cannot verify and cannot disprove. Students need a mental model of why the system cannot verify a step, so they can change the step accordingly. Making the reasoning system understandable is especially challenging. In the proof exercises described in this paper, we use a very simple model of “triviality”: there is a set of trivial rules. But even with this simple model, the sheer number of rules means that it is difficult for a student to know whether or not a proof step requires a non-trivial rule. With a more complex notion of triviality, it might become very difficult indeed for a user to understand why a step is not trivial to the reasoning system.

9.6 Detailed Qualitative Results

The table below presents the collected issues from the user study and Cognitive Dimensions questionnaires. For ease of presentation, I have also listed the issues and feedback from the users against relevant Cognitive Dimensions. (The user comments were assigned to appropriate dimensions by me and informally checked by Alan Blackwell, an expert in CDs.)

Statements for issues that were first raised by a user are marked with a U. Statements that were not reported by users, but describe issues that were later revealed in the Cognitive Dimensions analysis are marked CD.

Premature Commitment		
1	U	The choice of which tiles to give the student often forced a single solution method on the student. For question 4, a student commented that they could have easily answered the question using the technique from question 3, but they had not been given the necessary tiles to do so.
2	U	The questions offered only provided tiles support for forward proof (moving forward from the premises, rather than backward from the goal).
Hidden Dependencies		
3	CD	A change made to an early line of proof can cause following proof commands that had worked before to fail. This was particularly noticeable where small algebraic changes are made (swapping a few terms around) that cause a rule that the simplifier used to no longer match the line.
4	U	Users did not understand that the `Rewrite` *`statement`* `for` *`expression`* command (that corresponds to Isabelle's `of[ ... ]` syntax) only works if the statement has a universally quantified variable in it. (Effectively, it only works if there is a '$\forall$' in the expression). Otherwise Isabelle/HOL marks the command with an error.
Viscosity		
5	U	Currently only free-standing tiles and nests of tiles can be copied or deleted. A user asked for a way to copy a tile that is in a socket without pulling it out of the socket first.
6	U	Expanding socket lists for commands only ensure that there is an empty socket at the end of the list. This leaves users having to manually shuffle commands down the list if they wish to insert a command in the middle.
7	CD	Although structural changes (eg, swapping two nests of tiles) can be very fast, some other actions are slower than if textual edits were allowed – for example changing $(a + b) \times c$ to $(b + c) \times a$.
Visibility		
8	U	If a line of proof is particularly long, the `by simplification` can be hidden by the annotation pane (it can be revealed by scrolling). However, this means that if the line of proof fails, the error icon that is placed over `by simplification` is not immediately visible. This sometimes caused students not to realise that there was an error in their proof, and they would become confused as to how come the congratulatory message saying they had completed the proof did not appear.
9	U	For one user, the bottom of the MathsTiles canvas happened to coincide with the bottom of his browser window, and it took some time for him to realise he needed to scroll down to find the *Check proof* link and other action links.

10	U	The tiles make the structure of the proof clear.
11	U	When tiles are added, they are always placed in the centre of the canvas and can hide each other.
Closeness of Mapping		
12	CD	The text of the tiles does more closely map a written proof than Isar syntax. However, a student would not normally write *“by simplification”* at the end of each line.
Consistency		
13	U	A user working through the introduction question was confused that the Expression button could not generate the *Proposition for ...* tile even though it is used in expressions. She tried a number of different ways of typing it before emailing for help and did not notice that there was a “Proposition for ...” button she could use to generate this tile. (The expression button was removed from the introduction question, but left in later questions – see Section 9.3.1 for discussion of this).
14	U	A number of users became confused that when they tried to prove a line, Isabelle/HOL did not remember all the lines of proof that have gone before, but only the assumptions and the immediately previous line. *“I established $A = B$ and $B = C$ after a number of steps each, but when I then want to show that $A = C$ the state space appears to forget that $A = B$.”* (If a user needed to use an earlier line as a premise much later, they needed to label it and then re-introduce it with “$\because$ `with` *`label`* `we have ...`”).
15	U	Because the selection of hint and advice functions did not remember which functions this student had already used, the same hint function could be selected a second time before all the other hints had been tried. Users then appeared to assume that there were no other hint functions available.
16	CD	If there are three comments on a tile, three comment icons are shown rather than one.
17	CD	The *Statement Label* control in the tile tray is inconsistent – when a new label is entered, it produces a label with a socket; when a label is repeated it produces a label reference with no socket. This is particularly inconsistent because a reference is of type *rule* whereas a label is of type *expression* but the control is always listed in the expression section of the tile tray.
Diffuseness		
18	U	One user expressed particular appreciation for the Expression button because it is much faster to type simple expressions where the syntax is obvious than to construct them with the mouse. (This user used the system after the Expression button was removed from the introduction, and so first used it in the second question).
19	U	*“More talented students may become frustrated at the lack of ‘obviousness’, for example explicitly having to use the $GCD(x, y) = GCD(y, x)$ tile”*
20	U	*“The system focusses on very formal proofs with only small steps allowed by Isabelle. This would be very useful for introducing first year undergraduates to formal proof. However, for teaching discrete maths I think it might distract attention from the core idea of the proof to getting all the fiddly details right”*

21	CD	The tiles for proof commands are necessarily more verbose to read than the Isar keywords they translate to. However, since proof commands are inserted using the mouse, the number of words on a tile does not affect the effort to insert a tile into the proof.
Error-proneness		
22	CD	When dragging a very large nest of tiles, it is easy to obscure the socket you want to drop it into and a number of other empty sockets as well, making it unclear where it will go.
Hard Mental Operations		
23	U	A bug in one question (later fixed) caused a proof line to fail because the mod tile incorrectly bracketed itself both visually and in the translation. Users were unable to determine that it was a bug, however, and when the proof line failed they wondered if there was a missing rule that they should have used. This suggests users find it quite hard to think critically about whether the system is operating as expected.
Progressive Evaluation		
24	CD	It is possible to check an incomplete proof and see whether or not the lines of proof so far have succeeded.
25	CD	The only measure of how near you are to completing the proof, however, is whether there are any useful rules for the question that you have not needed to use yet. (The prover does not know how to solve the question automatically, so there is no yardstick to measure against).
Provisionality		
26	CD	Because nests of tiles can be unplugged and left loose on the canvas (out of the proof but undeleted), it is relatively easy to de-commit from parts of the proof, sketch out, and change your mind.
Role-expressiveness		
27	U	A number of users were not aware that by typing a label you had already used into the Statement Label button, you would get a reference to that statement.
28	U	Users frequently used the Expression button to try to generate incomplete expression fragments to add to the canvas. (For example, just typing "$=$"). Some of these expression fragments were beyond the capability of the parser behind the Expression button to parse.
29	U	The explanation for one of the error messages was missing from the Book. When a user clicked the *"What does this mean?"* link for the error, the Book took its default action when it cannot find any entries for a topic of presenting a set of search results and links for adding your own entry into the Book. The user was surprised by the sudden appearance of a set of search results and thought that something on the server had broken.
30	CD	Clicking an annotation icon brings up details of all the annotations on that tile, not just the one you clicked on.

9.7 Comparison to ActiveMath

Of the related work described in Chapter 2, ActiveMath [MAB$^+$01, MBG$^+$03, MS04, LG06] is the most similar in aim. It is an ongoing project to develop interactive Web-based textbooks that combine both exercises and content. It is worth, therefore, describing the differences between my system and ActiveMath in more detail.

The major difference is that the Intelligent Book, described in this dissertation, takes a more informal approach to modelling both the content and the exercises. Content in ActiveMath is defined in the semantic OMDoc format [Koh00], and is not authorable by students. It is regarded as a canonical representation of mathematics, and personalised lessons are generated from it. The content in the Intelligent Book, meanwhile, takes a looser approach in which multiple entries for the same content can co-exist, and keeps a minimal amount of semantic information, to ensure that users can add content to the book without extensive training in its content model.

In the ActiveMath system, exercises are expected to provide detailed feedback to the student model, which rates students against competencies for each concept in the system. The Intelligent Book, meanwhile, does not specify a student model, but leaves it to question-type authors to decide what student modelling, if any, a question should perform. This is particularly designed to support Reactive Learning Environment questions, where it might not be feasible to model students. The proof exercises in this dissertation, for instance, might be difficult to integrate into ActiveMath because it is currently impossible to determine precisely *why* a student failed to complete a particular proof.

A third difference is that, at the time of writing, ActiveMath has not focussed on exercises that require a graphical interface and cannot be represented in text or HTML. The only graphical exercise I am aware of in ActiveMath is a modelling exercise in which students draw their own concept map for a topic, and this is compared to the concept map that can be derived from the OMDoc content [MKH05]. This exercise was, however, written and published some time after I developed the graphical exercise architecture described in Chapter 3.

However, there are ways in which the two projects have begun to look at similar issues. For example, Claus Zinn [Zin06] noted that Wiki content can be produced much more quickly than ActiveMath's carefully written semantic content, and the amount of mathematics Wiki content on the Web is growing much faster than the content within ActiveMath. He has therefore begun to examine ways in which the OMDoc content of ActiveMath could be used to provide seed content for a semantic Wiki, called se(ma)^{2}wi.

CHAPTER 10

Searching Questions

Although some users in the study in Chapter 9 successfully completed proofs, which is more than we could expect if they attempted proofs directly in Isabelle/HOL with so little training, the study revealed a number of significant usability problems. Some of these related to simple oversights in the MathsTiles applet that would be a straightforward coding exercise to fix (for example, the annotations often obscure some of the text on the tiles). Some of the usability issues, however, were more fundamental and relate to the fact that humans and automated systems have a very different notion of whether one line of symbolic proof trivially follows from another. For example, rearranging the order of an algebraic expression is often a trivial exercise for a human, but requires the combination of many different rules of algebra for Isabelle/HOL to check that it is correct. In fact there are approximately 1,500 rules that Isabelle/HOL's simplifier considers "trivial" in most questions, and this large number in turn makes it very difficult indeed for a student to know which rules Isabelle/HOL does not think are simple, and therefore must be mentioned explicitly in the proof.

In this chapter, based on observations of students attempting proofs in front of a human tutor, I examine whether an informal model might be able to support proof questions that can be made usable with much less effort. When generalised and simplified, I show how these search-based questions can also be used as a replacement for multiple choice questions, or to provide "massively multiple choice" questions.

10.1 Classroom Observations

In 2005, with the assistance of Kasim Rehman, I observed and video recorded a series of tutorial sessions in which students worked through homework exercises on the blackboard in front of their peers and a tutor as part of their Discrete Mathematics course. We recorded 13 sessions, with four students answering questions in each session, in front of one of four tutors.

Unsurprisingly, when students became stuck I observed that tutors would often try to guide them to the expected answer for the question, which was listed on an answer sheet held by the tutors. Surprisingly, however, I also observed occasions where the student found an unanticipated solution to the exercise (which was accepted) but the tutor still felt the need to explain what the expected solution on the answer sheet had been. This suggested that perhaps the ideal of giving equal support to every possible solution in an Intelligent Book exercise is unnecessary. Even human tutors, often found to be the ideal teaching scenario [Blo84, KK91], sometimes focus on an expected solution. This might in fact be the correct strategy – homework exercises are not usually set for the sheer beauty of setting a question, but to give the student experience in a taught area. Indeed if the question setter did not have a solution in mind, how would he or she have known that it was a reasonable question to set?

I also observed that many of the questions set in the mathematics course do not call for an answer phrased as a symbolic proof, but a more informal English language argument. For example, consider the following two questions from the tutorial sessions, together with their expected answers. (These answers have been rephrased slightly to make them more readable for this dissertation.)

1. Show that the set of irrational numbers is uncountable.

   (a) We suppose that the set of irrational numbers, $\mathbb{I}$, is countable and derive a contradiction. Suppose that $\mathbb{I}$ is countable. Every real number is either rational or irrational. That is, $\mathbb{R} = \mathbb{Q} \cup \mathbb{I}$. The set of rational numbers, $\mathbb{Q}$, is countable. The union of two countable sets is countable. So the set of real numbers, $\mathbb{R}$, must be countable. But $\mathbb{R}$ is uncountable – a contradiction.

2. Show that any set of disjoint discs (ie, circular areas which may or may not include their perimeters and that do not overlap) in the plane (a two-dimensional plane) is countable. You may assume that the rational numbers are dense in the real numbers, in the sense that for any reals , there is a rational such that .

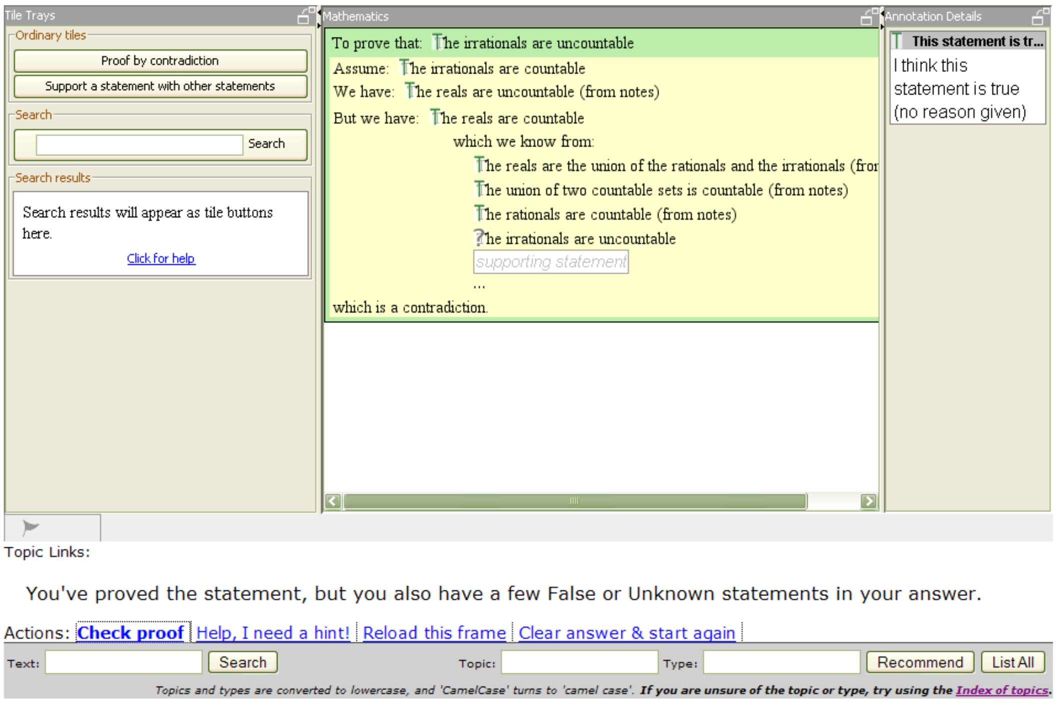
   (a) Let D be a collection of discs in the plane. For every disk, we can draw a hypothetical square aligned with the x and y axis, such that the corners of the square lie on the circumference of the circle. This square has corners at $(x_1, y_1), (x_2, y_1), (x_2, y_2)$, and (x_1, y_2). Since the rational numbers are dense in the real numbers, we have a rational number $q_d \in \mathbb{Q}$ such that $x_1 < q_d < x_2$, and a rational number $q'_d \in \mathbb{Q}$ such that $y_1 < q'_d < y_2$. The point (q_d, q'_d) certainly lies in the disc. We now define a function $g : D \longrightarrow \mathbb{Q} \times \mathbb{Q}$ as follows: $g(d) = (q_d, q'_d)$. Since the discs in D are disjoint, this function is an injection. Since $\mathbb{Q} \times \mathbb{Q}$ is countable and there is an injection from D to $\mathbb{Q} \times \mathbb{Q}$, D must also be countable.

Looking at these questions, there is little advantage to be gained from modelling the mathematics formally. We already know that the arguments, when constructed correctly, are formally true or otherwise we would not have set them as questions. So we are essentially looking for the students to say particular expected statements in the appropriate argumentative construct. The text of the second question appears more complex, and contains algebraic inequalities that look as if they could be modelled symbolically, but that would not be helpful. The inequalities are not used in any algebraic operations, but only to argue that because there are two distinct real coordinates and two distinct real coordinates on the disc, there must be a point with rational coordinates somewhere between them. (In fact, the original model answer had a slight mistake in the inequalities that went uncorrected for two years – this highlights that the algebra of the inequalities is not considered to be the important teaching point of the question.) Concepts such as drawing a square on the circle are awkward to model formally, but very easy to model informally as statements the student might say for this question.

For the tutorial sessions, the course planners explicitly asked students to explain the outline of their solution rather than focus on the specific algebra, but I also observed many similar questions in the course notes.

10.2 The Informally Modelled Scenario

I built a system for asking these kinds of questions that uses an informal modelling system with the same MathsTiles front end as I had used in the formally modelled questions. Tiles are provided for prewritten statements that the student might wish to use in his or her answer. A screenshot of a question is shown in Figure 10.1.



Tile Trays

Ordinary tiles

Proof by contradiction

Support a statement with other statements

Search

Search

Search results

Search results will appear as tile buttons here.

Click for help

Mathematics

To prove that: The irrationals are uncountable

Assume: The irrationals are countable

We have: The reals are uncountable (from notes)

But we have: The reals are countable

which we know from:

The reals are the union of the rationals and the irrationals (fro

The union of two countable sets is countable (from notes)

The rationals are countable (from notes)

?he irrationals are uncountable

supporting statement

...

which is a contradiction.

Annotation Details

This statement is tr...

I think this statement is true (no reason given)

Topic Links:

You've proved the statement, but you also have a few False or Unknown statements in your answer.

Actions: Check proof | Help, I need a hint! | Reload this frame | Clear answer & start again

Text: Search Topic: Type: Recommend List All

Topics and types are converted to lowercase, and 'CamelCase' turns to 'camel case'. ***If you are unsure of the topic or type, try using the Index of topics.***

Figure 10.1: An informal proof question. Students argue using predefined statements that they must find using search functions. (The feedback in this screenshot suggests that the student's answer contains statements and an argument that could prove the proposition, but there is an unproved and unnecessary statement in the argument that should be removed before the answer is correct. The unproved statement is indicated with a question mark.)

If a list of the possible statements was made available to the students then the exercise would change from requiring recall to only requiring recognition. Students, rather than having to think of the statements they need to use in their argument, would merely have to recognise them from the list. Furthermore, students would be able to solve the question by simple trial and error – trying out different combinations of the available statements until the system was happy with the answer. To avoid this, the interface does not show the list of statements that can be used in the question. Instead it requires students to search for their statements, forcing them to show they know something about the statements they wish to use. The search box is towards the left of the screen in Figure 10.1.

The search typed in by the student is required to contain a minimum number of keywords (normally two), and only tiles matching all the keywords in the search will be returned. The reason for this is to prevent very simple searches based on keywords in the question. For instance, if the question allowed searches on a single term, then it would be possible for students to search for all the available statements about the real numbers, or all the statements including the word countable. Requiring multiple terms makes this strategy less effective – statements often link concepts (eg, “the union of two countable sets is countable”), and if keywords are required for each concept then the student has to initiate the link between concepts, rather than finding linking statements in the list by accident.

The model used to keep track of the argument is a truth map stack. Each map in the stack maps statement IDs to either of the states true or false, and also remembers the reason why each statement is mapped to each state. A request for the truth of a statement will look for the most recent map containing that statement ID and return the associated state. Statements that are not in any of the maps are unknown. Maintaining a stack of maps provides a simple way for us to make temporary assumptions and reason about them. For example, in a proof by contradiction, we push a new map onto the stack and make the temporary assumption that a statement is true. Based on this assumption, we then prove further statements to be true, until we find a contradiction that shows our original assumption must have been mistaken. At that point, we discard the top map from the stack, that contains our assumption and all the temporary conclusions we drew from it, and mark the original statement as being false in the map underneath. The truth map stack is illustrated in Figure 10.2.

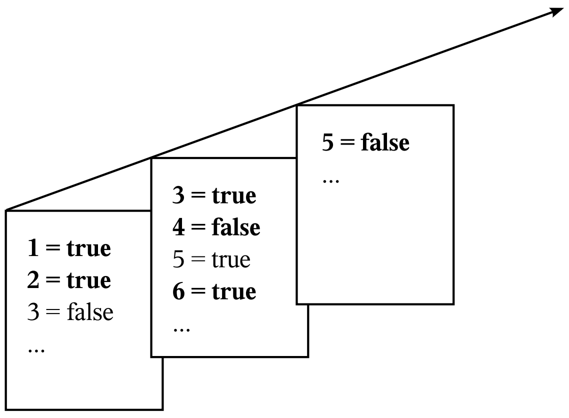


Figure 10.2: A truth map stack. Each level contains mappings from statement IDs to the states true or false. Maps can be added to the stack to temporarily override the existing mappings. Each mapping also holds a reference to the tile from the argument that caused it to be set (not shown).

The model is not driven by any automatic reasoning system, but by the argument that the student has written. The argument, as written in the tile language, forms a hierarchy of elements. Just as in the formal proof case, conversion scripts worked through the hierarchy to convert it into an Isar proof, so in this case conversion scripts work through the hierarchy. However, the output of these conversion scripts is not a document in another language, but a series of actions on the model. So for example, the matcher for contradiction tiles pushes the new truth map onto the stack, sets the statement in the tileâĂŹs assumption socket to be true, and tests for a contradiction in its other sockets. In the tile language for these questions, users can only assume or conclude that statements are true. (False statements are dealt with by assuming or concluding that the opposite statement is true: users cannot argue that “X” is false but must argue that “not X” is true.) The test for a contradiction, then, is to find two opposite statements that are both true.

Writing a question involves writing the statements that the student can use, specifying their keywords, marking which statements are opposite to which other statements, and defining a set of implication rules. The implication rules state that a statement is true (or false) if a list of other statements is true. Implication rules can set statements to be false even though students can only argue that statements are true. The reason for this is to allow the list of statements returned by a search to include statements that are incorrect.

These questions essentially use predicate logic to model the argument, and use pre-written statements for anything that requires a more complex logic. The system is, however, extensible beyond predicate logic – questions can include their own tiles and extend the conversion script to include their own matchers that implement the necessary checks.

10.3 Massively Multiple Choice Questions

In the previous section, I discussed questions where students have to search for statements to construct an argument. In this section I briefly consider how this applies to questions where students have to search for a single statement.

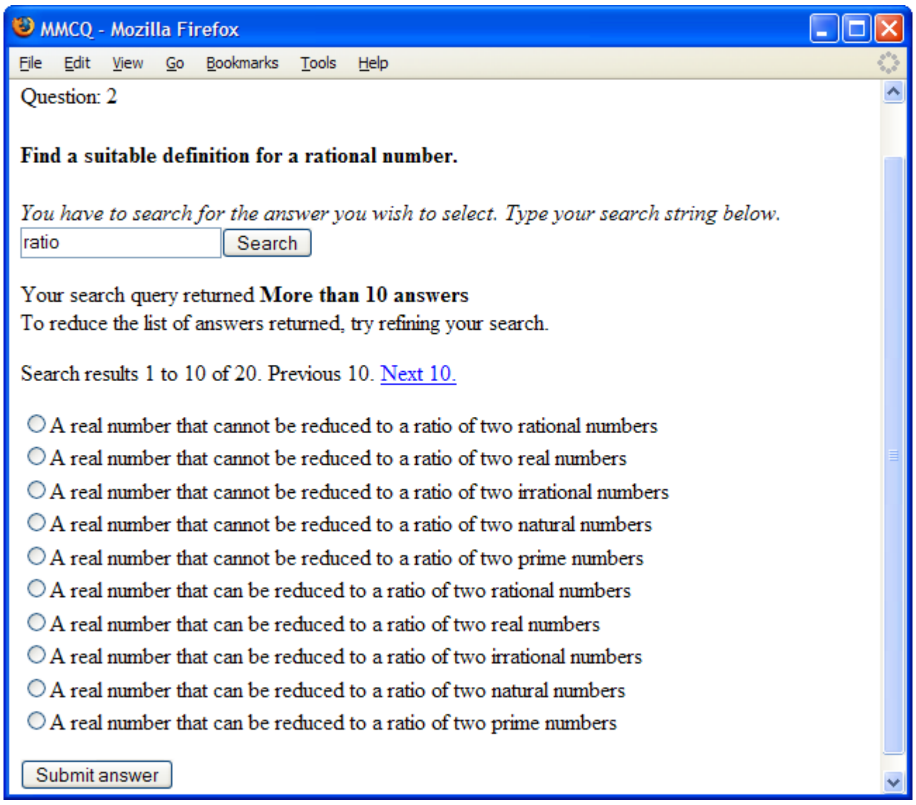
Prewritten statements have the advantage over asking students to write their own statements that they do not need any complex parsing or checking. The “searching for statements” paradigm was introduced so that students would not be able to recognise and select statements to use from a short list. It is possible, then, to consider “searching for statements” as a compromise between the short answer and multiple choice formats. The number of options can be much larger than is practical in traditional multiple choice because the options do not all need to be shown at the same time, but is not the theoretically infinite number of choices that the short answer format gives. For this reason, these can be considered to be *massively multiple choice questions* (MMCQs). I constructed a simple system for MMCQs, a screenshot of which is shown in Figure 10.3.

Again, these questions use the principle that knowing the probable answers in advance allows us to model the question more loosely. A more traditional approach would be to ask the student to enter a short answer and use Natural Language Processing (NLP) to analyse the answer. In this case because we already know what the student is likely to say, we effectively replace complex NLP with a simple keyword search and confirmation step.

The main advantage over traditional multiple choice questions is that the list of answers, being hidden, does not act as a prop. For example, consider the following mathematical puzzle (again from the Discrete Mathematics course) that does not work as a multiple choice question:

1. A prison houses 100 inmates, one in each of 100 cells, guarded by a total of 100 warders. One evening, all the cells are locked and the keys left in the locks. As the first warder leaves, she turns every key, unlocking all the doors. The second warder turns every second key, relocking every even numbered cell. The third warder turns every third key, and so on. Finally the last warder turns just the key in the last cell. Which doors are left unlocked and why?

   (a) The key to cell number n has been turned once for every factor of n. So the doors left unlocked are those with an odd number of factors.



Question: 2

Find a suitable definition for a rational number.

You have to search for the answer you wish to select. Type your search string below.

ratio [Search]

Your search query returned **More than 10 answers**
To reduce the list of answers returned, try refining your search.

Search results 1 to 10 of 20. Previous 10. Next 10.

- ○ A real number that cannot be reduced to a ratio of two rational numbers
- ○ A real number that cannot be reduced to a ratio of two real numbers
- ○ A real number that cannot be reduced to a ratio of two irrational numbers
- ○ A real number that cannot be reduced to a ratio of two natural numbers
- ○ A real number that cannot be reduced to a ratio of two prime numbers
- ○ A real number that can be reduced to a ratio of two rational numbers
- ○ A real number that can be reduced to a ratio of two real numbers
- ○ A real number that can be reduced to a ratio of two irrational numbers
- ○ A real number that can be reduced to a ratio of two natural numbers
- ○ A real number that can be reduced to a ratio of two prime numbers

[Submit answer]

Figure 10.3: A “massively multiple choice question” – the student is required to enter a search to return potential answers, and then select an answer from the resulting list. The student has searched for “ratio”, so only answers including the word “ratio” are returned. (In this particular question, students are not required to search for all or a minimum number of terms in the intended answer.)

If the answer is visible on the page then respondents are likely to pick it whether or not they had thought of it before. If respondents must search for the keywords “factor” and “odd” before that answer becomes visible, however, then that would reasonably restrict that answer to only those who had already thought of it. Similarly, in survey questions hiding the potential answers may prevent respondents from being distracted from their original answers. It remains open to argument, however, whether this is a benefit or not – whether a response from someone who has not seen the alternatives is “a less well-considered answer” or “unaffected by suggestion”. Nonetheless, just as there have been observed differences in studentsâĂŹ responses to multiple choice questions compared to short answer questions [PGWP90], I expect students would respond slightly differently again to these questions.

10.4 Conclusion

Neither the formal nor the informal system is obviously superior to the other, but they serve complementary roles in teaching mathematics, because the two systems allow us to ask different kinds of questions. In the formal system the questions were very symbolic, such as an induction proof of some algebraic statement on the Fibonacci sequence. These would be less well suited to the informal system because so many of the lines of the proof are algebra, and there is not a mechanism to search for algebra yet (although one could be imagined). The informal questions, as described before, focus on proofs where the argument is expressed in words. This makes it impractical to directly compare the usability of the two systems.

However, it is not simply the usability of the informally modelled questions that is their advantage – it is that it takes so much less development effort to produce a usable question. The conversion scripts and processing for the formally modelled system took several months of effort, and the Isabelle/HOL automated theorem prover that it uses no doubt took many PhD's worth of work for its developers to build. The informally modelled questions, meanwhile, were constructed over two afternoons, including their model.

There also appear to be two other advantages to a system using pre-planned answers:

- Students of mathematics take some time to become fluent in the formal language required for proofs. Allowing them to choose between syntactically correct but semantically different answers reinforces correct use of the language.
- Limiting the student to pre-planned answers using a simple model might also have another practical benefit. Often while there may theoretically be many routes to a proof, in a formal reasoning system there can be subtle reasons why some of the routes are very difficult to achieve. In a formally reasoned setting, students might spend a great deal of time trying a theoretically possible but practically unachievable route to a proof. In a limited and informally reasoned setting, they will perhaps be readier to decide that a route is not supported and try another more successful strategy. Of course, further research would be required to verify this hypothesis.

Generalising the "searching for statements" mechanic to allow massively multiple choice questions is an obvious extension of the questions I developed. In pencil-and-paper multiple choice tests, there is a clear technical need for the options to all be shown to the student at the start. However, in the client-server situation that has been common in online learning for many years now, there appears to be no need to give away the answer in the question, nor to limit the possible answers to only four or five.

CHAPTER 11

Conclusion

11.1 Summary of Contributions

This dissertation has presented the following contributions:

1. **Formally modelled exercises supporting student-written proofs in Number Theory**.
   Although there are many usability issues still to overcome, the exercises described in Chapters 6 to 9 represent an advance in enabling untrained students to write verifiable proofs in a system where the student must write the lines of proof (rather than asking an automated proof assistant to apply tactics to manipulate goal statements). There are many systems that ask students to write simple proofs in simpler domains such as predicate logic [LY02, LLB02], but this is the first Web-based learning environment to ask students to write proofs in this manner for Number Theory. The qualitative usability study revealed a number of issues that are relevant for future work on educational proof interfaces, as described in Chapter 9.

2. **A novel kind of structured interaction language**.
   As described in Chapter 2, structured editing is an established technique but MathsTiles is different in three ways. Firstly, it allows multiple code fragments to be scattered across the canvas, which means it does not have the restriction that "if it is on the page, it is in the code" that is common to other structured editors. Secondly, it is a structured editor for informally defined languages that translate to formal language, rather than for languages with formally defined syntaxes (and it allows students to make mistakes). Thirdly, it allows the interaction behaviour to be altered for individual pieces of syntax at run-time. For example, the green question tiles are individually set to be unselectable and indelible. A change message from the server, however, can remove that restriction, or make any other tile on the page unselectable. Another change message could introduce a new tile with a new tile definition, effectively altering the syntax of the language.

3. **Informally Modelled and "Searching" Questions**.
   The informally modelled questions, described in Chapter 10, introduce the concept of

"searching for statements". This allows complex questions to be modelled using much less complex reasoning: the parts that are complex to model are replaced by prewritten statements and search and select steps. The example showed how an argumentative proof in Number Theory could be modelled using predicate logic. This allowed development time for the question to be many orders of magnitude faster than the formally modelled Number Theory questions.

4. **Massively Multiple Choice Questions**.
   The massively multiple choice questions are a logical consequence of the informally modelled questions: they are the case where students are asked to search for a single statement. However, their wide applicability means they are worth discussing explicitly. They provide a means for asking multiple choice questions without acting as a prop for the student, and they support a very large selection of different answers without the natural language processing requirement of the short answer format.

5. **A novel architecture for an Intelligent Book**.
   Chapters 3 to 5 described an architecture for Intelligent Books. It supports questions where students work in graphical notations appropriate to the domain, and allows the teaching script to make comments as students work, rather than waiting for a submit button to be pressed. It supports different models, pedagogies, and graphical notations for different questions. Its content model is designed to be appropriate both for students and for the modelling or reasoning system that supports questions: students can add new content or alter existing content, and the system can automatically generate references to the content. Both the content architecture and the question architecture are designed to be more flexible and informal than in the most relevant other Web-based textbook project (see Section 9.7).

11.2 Future Directions

11.2.1 Improvements to MathsTiles

As noted in Section 7.7, MathsTiles as presented in Chapter 7 does not support editing by typing, even though this has been found to be useful. Adding this support would involve converting one-dimensional typed text into an *ad hoc* two-dimensional syntax, ideally without needing to teach the one-dimensional syntax explicitly to users.

In Section 9.4.3, I described how there are dependencies between elements in a proof, and proposed that the system should be able to adjust students' proofs automatically to maintain the dependencies. Determining parts of a tile from an expression in the tile definition could maintain the simplest dependencies. (In Section 7.7, I describe how it may be useful to introduce a general purpose expression language into tile definitions.) Additional modifications would also need to be made by logic in the Teaching Script for more complex dependencies. This raises the wider research question of how collaborative authorship of documents should be supported when the participants are a human and a reasoning system, rather than two humans.

11.2.2 Levels of Formality

In Section 9.4.4 I proposed that being able to configure the level of formality of the prover would be helpful. (For example, the prover could handle universal quantification differently depending on whether or not students have been taught the concept.) This would also support an "engineering" approach to questions. In Section 2.2.4, I described how engineers often work out a rough solution to a problem that they later refine. This is increasingly also true of mathematicians working with formal proof systems, through the use of proof planners and proof sketches to develop a formally verifiable proof. In Chapter 10 I introduced questions that use informal modelling. It might therefore be useful to support a transition from informal to formal models.

11.2.3 Further support for cases where the reasoning system is unsure

In the electronics question in Chapter 3, I described a technique for relating a automatically generated reasoning to a student's level of detail. In the formally modelled proof exercises in Chapter 9, however, one of the major difficulties was what to do when the reasoning system is unsure whether a statement is correct or not – there is no successful chain of reasoning to explain.

The questions in this dissertation used a simple technique of providing a selection of *ad hoc* advice functions that could, for example, try different numbers with an equation to see if it failed or provide potted advice written by the teacher. While these can be helpful, there are many well-known analysis techniques that were not used and should be in future versions. For example, although the simplifier was limited to only using the `simp` automated proof method, there is no reason why the advice functions should not use other proof methods, such as `blast` and `auto`. This would uncouple the concepts of whether a proof step is *provably true* and whether it is *acceptable* in a student's answer.

Dixon and Fleuriot [DF05] describe how in professional practice it can be more useful to use weaker proof methods that leave a readable proof state with some kind of progress, rather than stronger tools that either succeed or fail without helping the user. The progress from these "well-behaved" methods could equally help students to understand what the system can and cannot verify, as well as helping students to explore the proof. Meier and Melis [MM05] describe how meta-reasoning about why a proof attempt failed can help automated proof systems choose the right strategy to use. This information would clearly also be helpful to students. There would, however, need to be careful consideration about whether the automated help could allow students to game the system.

This problem of uncertainty is also likely to occur in other design tasks. For example, in a programming exercise it can be difficult for a modelling or reasoning system to assess whether a piece of program is "on the right track" until it has been completed.

11.2.4 Programming interfaces

Some of the usability issues raised suggest that proving is more like programming than I had anticipated. For example, the need for automatic labelling of proof statements is similar to line numbering. The annotations appearing on the tiles were found to be problematic because they could be obscured by other tiles or the edge of the window, and so a more traditional gutter seems appropriate. However, there are also aspects of the proof exercises that may be applicable to programming environments. For example, it has been observed that programmers frequently find themselves substituting blocks of code between a set of alternatives [KAM05]. Being able to extract syntactic sections of code and leave them on the page but not in the code might be helpful.

11.3 The Future of Intelligent Teaching Assistants

Despite three decades of research into intelligent teaching assistants, most university courses do not use one. Practically, the most significant barrier to their use is that they are expensive to develop and maintain. Often, universities develop their own teaching assistant for a particular course, perhaps funded as part of a research project. This, though, means that the high cost of maintaining the system is set against the few students who take that particular course each year.

Industrially, there have been efforts to standardise learning objects so that they can be reused between courseware management systems. This allows the same objects to be used for many students across different universities. However, while this reduces the development and maintenance effort, the effort is still significant. Each time a new version of a courseware management system or a learning object is developed, there is maintenance work involved in upgrading the system at the university site. This work occurs at every university that is using the system, and each different kind of “intelligent question” is another part to maintain.

While universities might not want their courseware management systems externally hosted (subcontracting the management of students’ learning could be seen as subcontracting a university’s core business), they do not feel the same pressure to produce their own textbooks for every course they teach. An “Intelligent Publisher” could host Intelligent Books for a number of different universities, and could be responsible for developing new kinds of exercise. The books could be made to appear separate, so that for instance one university’s students do not see pages added by another’s, but as the system would be hosted by a single organisation, the exercise types could be reused between the textbooks with much less effort.

When the cost of developing and maintaining a question becomes less significant, many more techniques become possible. Questions could be developed that use many different modelling and reasoning systems, supporting the fact that humans often think about a problem on a number of levels. They would let students smoothly move from analysing a numerical example to describing what its implications are – for example, moving from calculating the capacity of the ocean to absorb carbon to discussing what that means for environmental policy. Questions could try to infer the student’s mental model of how concepts fit together, rather than only rating students against concept maps written by the teacher. Other questions could be integrated into real world systems – for example, traffic engineering questions that use a constantly up-to-date model of the country’s transport infrastructure.

Bibliography

[ABP$^+$04] Peter B Andrews, Chad E Brown, Frank Pfenning, Matthew Bishop, Sunil Issar, and Hongwei Xi. ETPS: A system to help students write formal proofs. *Journal of Automated Reasoning*, 32:75–92, 2004.

[ABY85] J. R. Anderson, C. F. Boyle, and G. Yost. The geometry tutor. In *Proceedings of IJCAI-85*, pages 1–7, 1985.

[ACKP95] John R. Anderson, A. T. Corbett, Kenneth R. Koedinger, and R. Pelletier. Cognitive tutors: lessons learned. *The Journal of the Learning Sciences*, 4(2):167 – 207, 1995.

[ACP01] A. Abel, B. Chang, and F. Pfenning. Human-readable machine-verifiable proofs for teaching constructive logic. In Uwe Egly, Armin Fiedler, Helmut Horacek, and Stephan Schmitt, editors, *Proceedings of the Workshop on Proof Transformations, Proof Presentations and Complexity of Proofs (PTP01)*, 2001.

[AG04] S. E. Ainsworth and S. K. Grimshaw. Evaluating the REDEEM authoring tool: Can teachers create effective learning environments? *International Journal of Artificial Intelligence in Education*, 14:279–312, 2004.

[AHW90] Farah Arefi, Charles E. Hughes, and David A. Workman. Automatically generating visual syntax-directed editors. *Commun. ACM*, 33(3):349–360, 1990.

[Ain06] Shaaron Ainsworth. 10 years, 30 learning environments, 850 students and one authoring tool: Lessons learned. Keynote speech to 6th IEEE International Conference on Advanced Learning Technologies, July 2006.

[ALW05] David Aspinall, Christoph Lüth, and Daniel Winterstein. Parsing, editing, proving: The pgip display protocol. In *International Workshop on User Interfaces for Theorem Provers 2005 (UITP'05)*, 2005.

[AMG$^+$03] S. E. Ainsworth, N. Major, S. K. Grimshaw, M. Hayes, J. D. Underwood, B. Williams, and D. J. Wood. REDEEM: Simple intelligent tutoring systems from usable tools. In T. Murray, S. Blessing, and S. Ainsworth, editors, *Advanced Tools for Advanced Learning Technology*, pages 205–232, Amsterdam, 2003. Kluwer Academic Publishers.

[AMSK06] Vincent Aleven, Bruce M McLaren, Jonathan Sewall, and Kenneth R Koedinger. The cognitive tutor authoring tools (ctat): Preliminary evaluation of efficiency gains. In *Proceedings of the 8th International Conference on Intelligent Tutoring Systems (ITS2006)*, pages 61–70, 2006.

[Ana83] J. Anania. The influence of instructional conditions on student learning and achievement. *Evaluation in Education: An International Review Series*, 7(1):1–92, 1983.

[And83] John R. Anderson. *The architecture of cognition*. Harvard University Press, 1983.

[And93] John R. Anderson. *Rules of the Mind*. Erlbaum, Hillsdale, NJ, 1993.

[And96] John R. Anderson. ACT: a simple theory of complex cognition. *American Psychologist*, 51:355–365, 1996.

[ASF99] Sherman R. Alpert, Mark K. Singley, and Peter G. Fairweather. Deploying intelligent tutors on the web: An architecture and an example. *International Journal of Artificial Intelligence in Education*, 10:183–197, 1999.

[AT00] Jonathan Allen and Christopher J. Terman. An interactive learning environment for VLSI design. *Proceedings of the IEEE*, 88(1):1–11, January 2000.

[BB04] William Billingsley and John Billingsley. The animation of simulations and tutorial clients for online teaching. In *Proceedings of the 15th Annual Conference for the Australasian Association for Engineering Education and the 10th Australasian Women in Engineering Forum, Toowoomba, Australia*, pages 532 – 540, 2004.

[BBB75] John Seely Brown, R R Burton, and A G Bell. SOPHIE: A step towards a reactive learning environment. *International Journal of Man-Machine Studies*, 7:675–696, 1975.

[BCF04] Peter Brusilovsky, Girish Chavan, and Rosta Farzan. Social adaptive navigation support for open corpus electronic textbooks. In *Adaptive Hypermedia 2004*, number 3137 in LNCS, pages 24–33, 2004.

[BCKW04] Ryan Shaun Baker, Albert T Corbett, Kenneth R. Koedinger, and Annette Wagner. Off-task behavior in the cognitive tutor classroom: when students "game the system". In Elizabeth Dykstra-Erickson and Manfred Tscheligi, editors, *Proceedings of ACM CHI 2004 Conference on Human Factors in Computing Systems*, pages 383–390, 2004.

[BE94] J Barwise and J Etchemendy. *Hyperproof*. CSLI Publications, Stanford, California, 1994.

[BFGHS04] Peter Baumgartner, Ulrich Furbach, Margret Groß-Hardt, and Alex Sinner. Living book - deduction, slicing, and interaction. *Journal of Automated Reasoning*, 32(3):259–286, 2004.

[BG03] Alan Blackwell and Thomas Green. Notational systems - the Cognitive Dimensions of Notations framework. In John M. Carroll, editor, *HCI Models, Theories and Frameworks*, pages 103 – 133, Amsterdam, 2003. Morgan Kaufmann.

[BG07] Alan Blackwell and Thomas Green. A cognitive dimensions questionnaire, 2007. Available online from http://www.cl.cam.ac.uk/ afb21/CognitiveDimensions/CDquestionnaire.pdf. Accessed 25 February 2007.

[BGHS02] Peter Baumgartner, Margaret Gross-Hardt, and Anna B Simon. Living Book - an interactive and personalized book. In Veljko Milutinovic, editor, *SS-GRR 2002s - International Conference on Advances in Infrastructure for e-Business, e-Education, e-Science, and e-Medicine on the Internet*, 2002.

[BHKK$^+$07] Christoph Benzmüller, Helmut Horacek, Ivana Kruijff-Korbayová, Manfred Pinkal, Jörg Siekmann, and Magdalena Wolska. Natural language dialog with a tutor system for mathematical proofs. *Journal of Computer Science and Technology*, 2007. To appear.

[BHL$^+$06] Christoph Benzmüller, Helmut Horacek, Henri Lesourd, Ivana Kruijff-Korbayova, Marvin Schiller, and Magdalena Wolska. A corpus of tutorial dialogs on theorem proving; the influence of the presentation of the study-material. In *Proceedings of International Conference on Language Resources and Evaluation (LREC 2006)*, Genova, Italy, 2006. ELDA. To appear.

[Bil01] John Billingsley. Javascript ‘Jollies’ can bring simulations to life. In *Proceedings of the 12th AAEE Conference on Engineering Education*, pages 63–67, Brisbane, Australia, 2001.

[BK04] Clemens Ballarin and Gerwin Klein. Introduction to the isabelle proof assistant. In *Second International Joint Conference on Automated Reasoning*, 2004. Available from http://isabelle.in.tum.de/coursematerial/IJCAR04/index.html. Accessed 24 February 2007.

[Blo56] Benjamin S. Bloom. *Taxonomy of Educational Objectives, Handbook I: The Cognitive Domain*. David McKay Co Inc, New York, 1956.

[Blo84] Benjamin Bloom. The two sigma problem: the search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, 13:4 – 15, 1984.

[BM04] Dave Beckett and Brian McBride, editors. *RDF/XML Syntax Specification (Revised)*. World Wide Web Consortium, 2004. Accessed 30 January 2005 http://www.w3.org/TR/2004/REC-rdf-syntax-grammar-20040210/.

[Bos01] Marat Boshernitsan. Harmonia: A flexible framework for constructing interactive language-based programming tools. Technical Report UCB/CSD-01-1149, University of California at Berkeley, 2001.

[BR05] William Billingsley and Peter Robinson. Towards an intelligent textbook for discrete mathematics. In *Proceedings of the 2005 International Conference on Active Media Technology, Takamatsu, Japan*, pages 291 – 296, 2005.

[BR07] William Billingsley and Peter Robinson. Student proof exercises using MathsTiles and Isabelle/HOL in an Intelligent Book. *Journal of Automated Reasoning*, 2007. in press.

[BRAH04] William Billingsley, Peter Robinson, Mark Ashdown, and Chris Hanson. Intelligent tutoring and supervised problem solving in the browser. In *Proceedings of the IADIS International Conference WWW/Internet 2004, Madrid, Spain*, pages 806 – 811, 2004.

[Bru96] Peter Brusilovsky. Methods and techniques of adaptive hypermedia. *User Modeling and User-Adapted Interaction*, 6(2–3):87–129, 1996.

[Bru00] Peter Brusilovsky. Adaptive hypermedia: from intelligent tutoring systems to web-based education. In *ITS2000*, LNCS, pages 1–7. Springer-Verlag, 2000.

[BRW96] P. Brusilovsky, S. Ritter, and G. Weber. ELM-ART: an intelligent tutoring system on the World Wide Web. In C. Frasson, G. Gauthier, and A. Lesgold, editors, *Lecture Notes in Computer Science*, pages 261–269. Springer-Verlag, 1996.

[Bur84] A. J. Burke. *Student's potential for learning contrasted under tutorial and group approaches to instruction*. PhD thesis, University of Chicago, 1984.

[Car70] Jaime Carbonell. AI in CAI: Artificial intelligence approach to computer aided instruction. *IEEE Transactions on Man-Machine Systems*, 11(4):190–202, 1970.

[CB97] A T Corbett and A Bhatnagar. Student modeling in the ACT programming tutor: Adjusting a procedural learning model with declarative knowledge. In *User Modeling: Proceedings of the Sixth International Conference UM97*, pages 243–254, 1997.

[CD99] James Clark and Steve DeRose, editors. *XML Path Language (XPath) Version 1.0*. World Wide Web Consortium, 1999. Accessed 30 January 2005 from http://www.w3.org/TR/1999/REC-xpath-19991116.

[CFR06] Don Chamerlin, Daniela Florescu, and Jonathan Robie, editors. *XQuery Update Facility, Working Draft 11 July 2006*. World Wide Web Consortium, 2006. Accessed from http://www.w3.org/TR/2006/WD-xqupdate-20060711/ on 13 Jan 2007.

[CGM$^+$04] R Conejo, E Guzman, E Millan, M Trella, J. L. Perez-de-la Cruz, and A. Rios. SIETTE: A web-based tool for adaptive testing. *International Journal of Artificial Intelligence in Education*, 14:29–61, 2004.

[CGV02] Christine Conati, A S Gertner, and Kurt VanLehn. Using Bayesian networks to manage uncertainty in student modeling. *User Modelling and User-Adapted Interaction*, 12(4):371–417, 2002.

[CHG$^+$00] Scotty D. Craig, Xiangen Hu, Barry Gholson, William Marks, Arthur C. Graesser, and The Tutoring Research Group. AutoTutor: A human tutoring simulation with an animated pedagogical interface. In P. Hamberger, editor, *Proceedings of the International Society for Optical Engineering: Integrated Command Environments*, SPIE Proceedings Series, 2000.

[CKK82] P. A. Cohen, C. C. Kulik, and J. A. Kulik. Educational outcomes of tutoring: a meta-analysis of findings. *American Educational Research Journal*, 19:237 – 248, 1982.

[Cla99] James Clark, editor. *XSL Transformations (XSLT) Version 1.0*. World Wide Web Consortium, 1999. Accessed 13 August 2006 from http://www.w3.org/TR/1999/REC-xslt-19991116.

[CT00] Albert Corbett and Holly Trask. Instructional interventions in computer-based tutoring: differential impact on learning time and accuracy. In *CHI '00: Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 97–104. ACM Press, 2000.

[DF05] Lucas Dixon and Jacques Fleuriot. A proof-centric approach to mathematical assistants. *Journal of Applied Logic*, 2005.

[EZ97] Jürgen Ehrensburger and Claus Zinn. DiaLog: A system for dialogue logic. In *Conference on Automated Deduction*, pages 446–460, 1997.

[FFJ03] Jon Ferraiolo, Jun Fujisawa, and Dean Jackson, editors. *Scalable Vector Graphics (SVG) 1.1 Specification*, chapter Paths. Word Wide Web Consortium, 2003.

[FMMCM04] Enrique Frías-Martínez, George Magoulas, Sherry Chen, and Robert Macredie. Recent soft computing approaches to user modeling in adaptive hypermedia. In *Adaptive Hypermedia 2004*, number 3137 in LNCS, pages 104–114. Springer-Verlag, 2004.

[For97] Kenneth D. Forbus. Using qualitative physics to create articulate educational software. *IEEE Expert*, 12(3), 1997.

[Gar05] Jesse James Garrett. Ajax: A new approach to web applications. Technical report, adaptivepath.com, 2005.

[GHJV95] Erich Gamma, Richard Helm, Ralph E. Johnson, and John Vlissides. *Design Patterns: elements of reusable object-oriented software*. Addison Wesley, 1995.

[GLJ$^+$04] A. C. Graesser, S. Lu, G. T. Jackson, H. Mitchell, M. Ventura, A. Olney, and M. M. Louwerse. AutoTutor: A tutor with dialogue in natural language. *Behavioural Research Methods, Instruments, and Computers*, 36:180–193, 2004.

[GP96] Thomas R G Green and M Petre. Usability analysis of visual programming environments. *Journal of Visual Languages and Computing*, 7, 1996.

[GRC02] E Guzman, J A Riveros, and R Conejo. A library for items construction in an adaptive evaluation system. In *Evidence Centred Design (ECD) Approach to Creating Diagnostic e-Assessments. San Sebastian*, pages 78–86, 2002. Also available online from http://www.lcc.uma.es/ accessed 1 March 2004.

[GV00] Abigail S Gertner and Kurt VanLehn. Andes: A coached problem solving environment for physics. In *Intelligent Tutoring Systems*, volume 1839 of *Lecture Notes in Computer Science*, pages 133–142, 2000.

[GWHWH$^+$99] Arthur C. Graesser, K. Wiemer-Hastings, P. Wiemer-Hastings, R. Kreuz, and The Tutoring Research Group. AutoTutor: a simulation of a human tutor. *Journal of Cognitive Systems Research*, 1:35–51, 1999.

[Han71] Wilfred J Hansen. Creation of hierarchic text with a computer display. Technical report, Argonne National Laboratories, 1971.

[HdAC$^+$04] V. Judson Harward, Jesús A. del Alamo, Vijay S. Choudary, Kimberley deLong, James L. Hardison, Steven R. Lerman, Jedidiah Northridge, Charuleka Varadharajan, Shaomin Wang, Karim Yehia, and David Zych. iLabs: A scalable architecture for sharing online experiments. In *International Conference on Engineering Education*, 2004.

[JJLZ04] Jr. Joseph J. LaViola and Robert C. Zeleznik. Mathpad2: a system for the creation and exploration of mathematical sketches. *ACM Trans. Graph.*, 23(3):432–440, 2004.

[JR99] Andreas Jochheim and Christof Röhrig. The Virtual Lab for teleoperated control of real experiments. In *Proceedings of the 38th IEEE Conference on Decision and Control*, 1999.

[KAH$^+$04] Kenneth R Koedinger, Vincent Aleven, Neil Heffernan, Bruce McLaren, and Matthew HockenBerry. Opening the door to non-programmers: Authoring intelligent tutoring systems by demonstration. In *Intelligent Tutoring Systems*, volume 3220/2004 of *LNCS*, pages 162–174, 2004.

[KAHM97] Kenneth R Koedinger, John R Anderson, William H Hadley, and Mary A Mark. Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, 8:30 – 43, 1997.

[KAM05] Andrew J. Ko, H. Aung, and Brad A. Myers. Eliciting design requirements for maintenance-oriented ides: A detailed study of corrective and perfective maintenance tasks. In *International Conference on Software Engineering*, 2005.

[Kay00] Judy Kay. Stereotypes, student models and scrutability. In *Intelligent Tutoring Systems*, number 1839 in LNCS, 2000.

[KCC$^+$02] Caitlin Kelleher, Dennis Cosgrove, David Culyba, Clifton Forlines, Jason Pratt, and Randy Pausch. Alice2: Programming without syntax errors. In *User Interface Software and Technology*, 2002.

[KK91] C C Kulik and J A Kulik. Effectiveness of computer based instruction: An updated analysis. *Computers in Human Behaviour*, 7:75 – 94, 1991.

[KKVdB00] S. Klai, T. Kolokolnikov, and N. Van den Bergh. Using Maple and the web to grade mathematics tests. In *Proceedings of the International Workshop on Advanced Learning Technologies, Palmerston, New Zealand*, 2000.

[KM06] Andrew J. Ko and Brad A. Myers. Barista: An implementation framework for enabling new tools, interaction techniques and views in code editors. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI2006)*, 2006.

[Koh00] Michael Kohlhase. OMDoc: Towards an internet standard for the administration, distribution and teaching of mathematical knowledge. In *AISC 2000 Artificial Intelligence and Symbolic Computation Theory*, pages 32–52, 2000.

[KSF99] Kenneth R. Koedinger, Daniel D. Suthers, and Kenneth D. Forbus. Component-based construction of a science learning space. *International Journal of Artificial Intelligence in Education*, 10:292–313, 1999.

[KU93] Amir Ali Khwaja and Joseph E. Urban. Syntax-directed editing environments: issues and features. In *SAC '93: Proceedings of the 1993 ACM/SIGAPP symposium on Applied computing*, pages 230–237, New York, NY, USA, 1993. ACM Press.

[KWR05] Viswanathan Kodaganallur, Rob R Weitz, and David Rosenthal. A comparison of model-tracing and constraint-based intelligent tutoring paradigms. *International Journal of Artificial Intelligence in Education*, 15:117–144, 2005.

[KWR06] Viswanathan Kodaganallur, Rob R Weitz, and David Rosenthal. An assessment of constraint-based turos: A response to mitrovic and ohlsson's critique of "A comparison of model-tracing and constraint-based intelligent tutoring paradigms". *International Journal of Artificial Intelligence in Education*, 16:219–321, 2006.

[Lam95] Leslie Lamport. How to write a proof. *The American Mathematical Monthly*, 102(7):600–608, 1995.

[LCL$^+$04] Guillaume Laforge, Clinton L Combs, Derek Lane, Chris Poirier, James Strachan, R L Winder, Boeing, and Thoughtworks. The Groovy programming language. Java Specification Request 241, Java Community Process, 2004.

[LEM04] Evelyn Lulis, Martha Evens, and Joel Michael. Implementing analogies in an electronic tutoring system. In *Intelligent Tutoring Systems*, LNCS, 2004.

[Ley83] F. S. Leyton. *The extent to which group instruction supplemented by mastery of the initial cognitive prerequisites approximates the learning effectiveness of one-to-one tutorial methods.* PhD thesis, University of Chicago, 1983.

[LG06] P. Libbrecht and C. Gross. Experience report writing leactivemath calculus. In William Farmer Jon Borwein, editor, *Proceedings of Mathematical Knowledge Management 2006*, number 4108 in LNAI. Springer Verlag, aug 2006.

[LLB02] Stacey Lukins, Alan Levicki, and Jennifer Burg. A tutorial program for propositional logic with human/computer interactive learning. In *Proceedings of the 33rd SIGCSE Technical Symposium on Computer Science Education*, pages 381–385. ACM Press, 2002.

[LM00] Andreas Laux and Lars Martin. *XUpdate – XML Update Language, Working Draft 2000-09-14.* XML:DB Initiative, 2000.

[LP00] Thomas Lozano Perez. Technologically enhanced education in electrical engineering and computer science, 2000. http://www.swiss.ai.mit.edu/projects/icampus/projects/eecs.html. MIT 6.001 tutor homepage. Last viewed December 2003.

[LY02] Leanna Lesa and Kalina Yacef. An intelligent teaching system for logic. In *Intelligent Tutoring Systems : 6th International Conference, ITS 2002, Biarritz, France and San Sebastian, Spain, June 2-7, 2002*. Springer-Verlag, 2002.

[MAB$^+$01] Erica Melis, Eric Andrès, Jochen Büdenbender, Adrian Frischauf, George Goguadze, Paul Libbrecht, Martin Pollet, and Carsten Ullrich. ActiveMath: A generic and adaptive web-based learning environment. *International Journal of Artificial Intelligence in Education*, 12(4):385–407, 2001.

[Mas02] J. Masthoff. Automatic generation of a navigation structure for adaptive web-based instruction. In P. Brusilovsky, N. Henze, and E. Millan, editors, *Proceedings of the AH'2002 Workshop on Adaptive Systems for Web-Based Education, Malaga, Spain*, 2002.

[MBG$^+$03] Erica Melis, Jochen Büdenbender, George Goguadze, Paul Libbrecht, and Carsten Ullrich. Knowledge representation and management in activemath. *Annals of Mathematics and Artificial Intelligence*, 38:47–64, 2003.

[MH00] Antonija Mitrovic and K. Hausler. Porting SQL Tutor to the web. In *Proceedings of the International Workshop on Adaptive and Intelligent Web-Based Educational Systems, Montreal, Canada*, pages 37–44, 2000.

[Mit02] Antonija Mitrovic. NORMIT: A web-enabled tutor for database normalization. In *Proceedings of the Interational Conference on Computers in Education (ICCE) 2002*, pages 1276–1280, 2002.

[MJP$^+$97] Allen Munro, Mark C. Johnson, Quentin A. Pizzini, David S. Surmon, Douglas M. Towne, and James L. Wogulis. Authoring simulation-centered tutors with RIDES. *International Journal of Artificial Intelligence in Education*, 8:234–316, 1997.

[MKH05] E. Melis, P Kärger, and M. Homik. Interactive concept mapping in ActiveMath. In Jörg N. Haake, Ulrich Lucke, and Djamshid Tavangarian, editors, *Delfi 2005: 3. Deutsche eLearning Fachtagung Informatik*, volume 66 of *LNI*, pages 247–258, Rostock, Germany, September 2005.

[MKM03] Antonija Mitrovic, Kenneth R Koedinger, and Brent Martin. A comparative analysis of cognitive tutoring and constraint-based modeling. In *Proceedings of the Ninth International Conference on User Modelling (UM 2003)*, number 2702 in LNAI, pages 313–322. Springer-Verlag, 2003.

[MM01] M. Mayo and Antonija Mitrovic. Optimising ITS behavior with Bayesian networks and decision theory. *International Journal of Artificial Intelligence in Education*, 12:124–153, 2001.

[MM05] Andreas Meier and Erica Melis. Impasse-driven reasoning in proof planning. In *Proceedings of the Fourth International Conference on Mathematical Knowledge Management (MKM2005)*, 2005.

[MMSM01] Antonija Mitrovic, Michael Mayo, Pradmuditha Suraweera, and Brent Martin. Constraint-Based Tutors: A success story. In *Engineering of Intelligent Systems : 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEA/AIE 2001, Budapest, Hungary*, 2001.

[MN06] Hirokazu Murao and Yasuhito Nakano. BrEdiMa: Yet another Web-browser tool for editing mathematical expressions. In *Proceedings of MathUI 2006*, 2006.

[MO99] Antonija Mitrovic and Stellan Ohlsson. Evaluation of a constraint-based tutor for a database language. *International Journal of Artificial Intelligence in Education*, 10:238–256, 1999.

[MO06] Antonija Mitrovic and Stellan Ohlsson. A critique of kodaganallur, weitz and rosenthal, "A comparison of model-tracing and constraint-based intelligent tutoring paradigms". *International Journal of Artificial Intelligence in Education*, 16(3):277–289, 2006.

[MPMV94] Phillip Miller, John Pane, Glenn Meter, and Scott Vorthmann. Evolution of novice programming environments: the structure editors of Carnegie Mellon University. *Interactive Learning Environments*, 4(2):140–158, 1994.

[MS04] E. Melis and Jorg Siekmann. Activemath: An intelligent tutoring system for mathematics. In L. Rutkowski, J. Siekmann, R. Tadeusiewicz, and L.A. Zadeh, editors, *Seventh International Conference 'Artificial Intelligence and Soft Computing' (ICAISC)*, volume 3070 of *LNAI*, pages 91–101. Springer-Verlag, 2004.

[Nip03] Tobias Nipkow. Structured proofs in Isar/HOL. In H Geuvers and F. Wiedijk, editors, *Types for Proofs and Programs (TYPES 2002)*, LNCS, pages 259 – 278. Springer-Verlag, 2003. Also available online from http://www4.informatik.tu-muenchen.de/ nipkow/pubs/types02.pdf accessed 10 June 2005.

[Nip06] Tobias Nipkow. A compact introduction to isabelle/hol, 2006. Available from http://isabelle.in.tum.de/coursematerial/Shanghai06/index.html. Accessed 24 February 2007.

[NPW02] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL — A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[NPW05] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle's Logics: HOL*. Technische Universität München, 2005.

[Ohl92] Stellan Ohlson. Constraint-based student modeling. *Journal of Artificial Intelligence and Education*, 3(4):429–447, 1992.

[PGWP90] M. Pressley, E. S. Ghatala, V. Woloshyn, and J. Pirie. Sometimes adults miss the main ideas and do not realize it: Confidence in responses to short answer and multiple choice comprehension questions. *Reading Research Quarterly*, 25(3):232–249, 1990.

[Pre26] Sidney L. A. Pressey. A simple apparatus which gives tests and scores – and teaches. *School and Society*, 23:373–376, 1926.

[RB97] Dave Raggett and Davy Batsalle. Mathematics on the Web: The EzMath notation. http://www.w3.org/People/Raggett/EzMath/EzMathPaper.html, 1997. Accessed 11 April 2007.

[Rit97] Steven Ritter. PAT Online: a model-tracing tutor on the World-Wide-Web. In *Proceedings of the Workshop Intelligent Educational Systems on the World Wide Web, 8th World Conference of the AIED Society*, 1997.

[RJR$^+$] Carolyn P Rosé, Pamela Jordan, Michael Ringenberg, Stephanie Siler, Kurt VanLehn, and Anders Weinstein. Interactive conceptual tutoring in atlas-andes.

[RK97] S. Ritter and K. R. Koedinger. An architecture for plug-in tutoring agents. *Journal of Artificial Intelligence in Education*, 7:315–347, 1997.

[Rob04] T. J. Roberts. The virtual machines laboratory. *Australasian Journal of Engineering Education*, (1):1–15, 2004.

[Rob06] Peter Robinson. *Discrete Mathematics I*. University of Cambridge Computer Laboratory, 2006.

[SAC86] Brian M. Slator, Matthew P. Anderson, and Walt Conley. Pygmalion at the interface. *Communications of the ACM*, 29(7):599–604, 1986.

[SBC$^+$05] Konrad Slind, Steven Barrus, Seungkeol Choe, Chris Condrat, Jianjun Duan, Sivaram Gopalakrishnan, Aaron Knoll, Hiro Kuwahara, Guodong Li, Scott Little, Lei Liu, Steffanie Moore, Robert Palmer, Claurissa Tuttle, Sean Walton, Yu Yang, and Junxing Zhang. Teaching a hol course: Experience report. In Joe Hurd, Edward Smith, and Ashish Darbari, editors, *Theorem Proving in Higher Order Logics: Emerging Trends Proceedings*, number PRG-RR-05-02 in Oxford University Computing Laboratory Technical Report, pages 170–179, August 2005.

[SCL$^+$01] Daniel D. Suthers, John Connelly, Alan Lesgold, Massimo Paolucci, Eva Erdosne Toth, Joe Toth, and Arlene Weiner. Representational and advisory guidance for students learning scientific inquiry. In K. Forbus and P. Feltovich, editors, *Smart machines in education*, pages 7–35. AAAI/MIT Press, Menlo Park, CA, 2001.

[Sco96] N. W. Scott. *A Study of the Introduction of Educational Technology into a Course in Engineering Dynamics: Classroom environment and learning outcomes*. PhD thesis, University of Western Australia, 1996.

[SGK$^+$05] Anders Selmer, Mike Goodson, Markus Kraft, Siddhartha Sen, V. Faye McNeill, Barry S. Johnston, and Clark K. Colton. Performing process control experiments across the atlantic. *Chemical Engineering Education*, 9:232–237, 2005.

[SKCM06] Anders Selmer, Markus Kraft, Clark Colton, and Ralf Moros. Weblabs in chemical engineering education. Technical report, University of Cambridge, 2006.

[Ski54] B. F. Skinner. The science of learning and the art of teaching. *Harvard Educational Review*, pages 86–97, 1954.

[Ski58] B. F. Skinner. Teaching machines: From the experimental study of learning come devices which arrange optimal conditions for self-instruction. *Science*, 128:969–977, 1958.

[SM02] P. Suraweera and Antonija Mitrovic. KERMIT: a constraint-based tutor for database modeling. In P. Cerri, G. Gouarderes, and F. Paraguacu, editors, *Proceedings of the 6th International Conference on Intelligent Tutoring Systems ITS 2002, Biarritz, France*, pages 377–387, 2002.

[SN04] Richard Sommer and Gregory Nuckols. A proof environment for teaching mathematics. *Journal of Automated Reasoning*, 32:227 – 258, 2004.

[SP96] V. J. Shute and J. Psotka. Intelligent tutoring systems: Past, present and future. In D. Jonassen, editor, *Handbook of Research on Educational Communications and Technology*. Scholastic Publications, 1996.

[SS77] Richard Stallman and Gerald J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9:135–196, 1977.

[SS94] Richard Scheines and Wilfried Sieg. Computer environments for proof construction. *Interactive Learning Environments*, 4(2):159–169, 1994.

[SS98] N Scott and B Stone. A flexible web based tutorial system for engineering, maths and science. In D. Jonassen, editor, *Handbook of Research on Educational Communications and Technology*. Scholastic Publications, 1998.

[SST+01] R Shelby, K Schulze, D Treacy, M Wintersgill, Kurt VanLehn, and A Weinstein. An assessment of the Andes tutor. In *Proceedings of the Physics Education Research Conference, July 21-25, Rochester, NY*, 2001.

[Sut03] Daniel D. Suthers. Representational guidance for collaborative learning. In H. U. Hoppe, F. Verdejo, and Judy Kay, editors, *Artificial Intelligence in Education*, pages 3–10. IOS Press, Amsterdam, 2003.

[Ten82] G. Tenenbaum. *A method of group instruction which is as effective as one-to-one tutorial instruction.* PhD thesis, University of Chicago, 1982.

[TR81] Tim Teitelbaum and Thomas Reps. The Cornell Program Synthesizer: a syntax-directed programming environment. *Communications of the ACM*, 24(9):563–573, September 1981.

[VDH01] Joris Van Der Hoeven. GNU TeXmacs: a free, structured, wysiwyg and technical text editor. *Le document au XXI-ième siècle*, 39–40:39–50, May 2001.

[VF04] Sekhar Vajjhala and Joe Fialli, editors. *The Java Architecture for XML Binding (JAXB) 2.0 (early draft v0.4).* Sun Microsystems, Inc., 2004.

[VLS+05] K. VanLehn, C. Lynch, K. Schulze, J.A. Shapiro, R. Shelby, L. Taylor, D. Treacy, A. Weinstein, and M. Wintersgill. The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education*, 15:147–204, 2005.

[Wal84] H. J. Walberg. Improving the productivity of America's schools. *Educational Leadership*, 41(8):19–27, 1984.

[WB01] Gerhard Weber and Peter Brusilovsky. ELM-ART: An adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education*, 12:351–384, 2001.

[WBG04] Daniel Winterstein, Alan Bundy, and Corin Gurr. Dr. Doodle: a diagrammatic theorem prover. In *Second International Joint Conference, IJCAR 2004, Ireland, July 4-8,* Lecture Notes in Computer Science, pages 331–335. Springer, 2004.

[WBGJ02] Daniel Winterstein, Alan Bundy, Corin Gurr, and Mateja Jamnik. Using animation in diagrammatic theorem proving. In *Diagrams 2002*. Springer-Verlag, 2002.

[Wen99] Markus Wenzel. Isar - a generic interpretative approach to readable formal proof documents. In *Theorem Proving in Higher Order Logics, 12th International Conference, TPHOLs'99*, LNCS, 1999. Also available online from http://www4.in.tum.de/ wenzelm/papers/Isar-TPHOLs99.pdf accessed 10 June 2005.

[Wen05] Markus Wenzel. *The Isabelle/Isar Reference Manual*. TU München, 2005.

[Wie04] Freek Wiedijk. Formal proof sketches. In *Types for Proofs and Programs*, volume 3085/2004 of *LNCS*. Springer, 2004.

[Win99] David Winer. *XML-RPC Specification*. UserLand Software, 1999. Accessed 30 January 2005 from http://www.xmlrpc.com/spec.

[Yac03] Kalina Yacef. Experiment and evaluation results of the logic-ita. Technical Report 542, University of Sydney, 2003.

[Yac04] Kalina Yacef. Making large class teaching more adaptive with logic-ita. In *Proceedings of the sixth conference on Australasian computing education*, pages 343–347, 2004.

[Zin06] Claus Zinn. Bootstrapping a semantic Wiki application for learning mathematics. In S. Schaffert and Y. Sure, editors, *Semantic Systems: From Visions to Applications. Proc. of the Semantics 2006 Conference.*, pages 255–260. Austrian Computer Society, 2006.

APPENDIX A

Abstract Teaching Script for Formal Proof Exercises

An abstract class is usually written for a type of question, and the scripts for individual questions are subclasses of that Abstract Teaching Script. The code listing below is the abstract superclass for the formally modelled proof exercises.

```
/*
 * Created on Mar 9, 2005
 *
 */
package cam.cl.intelligentBook.proof;

import java.io.IOException;
import java.io.StringReader;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;

import javax.servlet.ServletException;

import cam.cl.intelligentBook.datalog.Datalog;
import cam.cl.intelligentBook.isabelleExpr.EvalException;
import cam.cl.intelligentBook.isabelleExpr.FunctionCallback;
import cam.cl.intelligentBook.isabelleExpr.ParseException;
import cam.cl.intelligentBook.isabelleExpr.SimpleNode;
import cam.cl.intelligentBook.isabelleExpr.TokenMgrError;
import cam.cl.intelligentBook.isabelleExpr.isar;
import cam.cl.intelligentBook.questions.DocumentKey;
import cam.cl.intelligentBook.questions.QuestionScriptException;
import cam.cl.intelligentBook.questions.TeachingScript;
import cam.cl.intelligentBook.questions.Util;
import cam.cl.intelligentBook.questions.XPathHandler;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Set;
import java.util.Vector;
import java.util.logging.Level;

import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

/**
 * Abstract teaching script for Isar/MathsTiles based proof questions
 *
 */
public abstract class ProofQuestionScript extends TeachingScript implements
    FunctionCallback {

  public static final String MATHSTILES_NAMESPACE = "http://www.cl.cam.ac.uk/
      users/whb21/MathsTiles";

  protected GroovyIsarProcessor isarProcessor;

  public ProofQuestionScript() throws ServletException, QuestionScriptException {
    super();
    isarProcessor = new GroovyIsarProcessor();
    this.xpathHandler = new XPathHandler(MATHSTILES_NAMESPACE);

    suggestFixAdviceMap.add(new String[] {"suggestFix_FailedToFinishProof", "
        relevance_FailedToFinishProof"});
    suggestFixAdviceMap.add(new String[] {"suggestFix_UnexpectedEndOfInput", "
        relevance_UnexpectedEndOfInput"});
    suggestFixAdviceMap.add(new String[] {"suggestFix_TileInIllegalLocation", "
        relevance_TileInIllegalLocation"});
    suggestFixAdviceMap.add(new String[] {"
        suggestFix_LocalStatementWillFailToSolveAnyPendingGoal", "
        relevance_LocalStatementWillFailToSolveAnyPendingGoal"});
    suggestFixAdviceMap.add(new String[] {"suggestFix_ProofCommandFailed", "
        relevance_ProofCommandFailed"});
    suggestFixAdviceMap.add(new String[] {"suggestFix_CannotRewriteStatement", "
        relevance_CannotRewriteStatement"});
  }

  /**
 * Checks the proof using Isar, using the default Conversion Script
 */
public Vector checkProof() throws ServletException {
  return checkProof("DefaultIsar.groovy");
}

/**
 * Checks the proof using Isar, using the specified Conversion Script
 */
public Vector checkProof(String scriptPath) throws ServletException {
  try {
     DocumentKey documentKey = new DocumentKey(this.getUserName(), this.
          getSubCollection(), this.getDocumentId());
     document = documentManager.getDocument(documentKey);
     Datalog.logSnapshot(this.getActionKey(), this.getUserName(), this.
          getDocumentId(), this.getSubCollection(), documentManager.
          getXmlContents(document));

     preChangeSetup();

     this.addCodedResponseCall("mundane", "clear annotations", "content.
          clearAnnotations");

     ProverResponseItem[] r_arr = isarProcessor.doIsar(document, scriptPath);

     DocumentKey annotationDK = new DocumentKey(documentKey.
          getUsername(), documentKey.getCollection(), "annotations_" +
          documentKey.getDocumentName());
     Document annotationDoc = documentManager.createDocument(annotationDK
          , null, "annotations");

     boolean foundError = false;
     for (ProverResponseItem pri : r_arr) {
       if (pri.xmlContextPath != null && pri.xmlContextPath.endsWith("/")) {
          pri.xmlContextPath = pri.xmlContextPath.substring(0, pri.xmlContextPath.
               length() - 1);
       }

       if (pri.responseLabel == "error" && !Util.empty(pri.xmlContextPath)) {
          // We stop showing errors after the first one, because they tend to be "follow
               -on" errors. Note, we only worry about errors with a response path, so
               we don't stop after errors in the header
          if (!foundError) {
             this.addResponseCall("content.annotate", pri.xmlContextPath, pri.
                  responseLabel, pri.responseTitle, pri.responseText, pri.responseCode)
                  ;
```

```
            foundError = true;
          }
        } else {
          this.addResponseCall("content.annotate", pri.xmlContextPath, pri.
              responseLabel, pri.responseTitle, pri.responseText, pri.responseCode);
        }

        /*
         * Add the response into the annotation document
         */
        if (!Util.empty(pri.responseLabel)) {
          Element e = annotationDoc.createElement(pri.responseLabel);
          e.setAttribute("xpath", pri.xmlContextPath);
          e.setAttribute("code", pri.responseCode);
          e.setAttribute("title", pri.responseTitle);
          if (pri.responseText != null && pri.responseText.length() > 0) {
            Element xmlE = annotationDoc.createElement("text");
            xmlE.appendChild(xpathHandler.parseXml(annotationDoc, Util.
                xmlEncode(pri.responseText), null));
            e.appendChild(xmlE);
          }
          if (pri.responseXml != null && pri.responseXml.length() > 0) {
            Element xmlE = annotationDoc.createElement("xml");
            xmlE.appendChild(xpathHandler.parseXml(annotationDoc, pri.
                responseXml, null));
            e.appendChild(xmlE);
          }
          annotationDoc.getDocumentElement().appendChild(e);
        } else {
          logger.warning("A prover response had an empty label (text follows): " + pri.
              responseText);
        }

      }

      documentManager.setModified(annotationDK);

      if (checkDone(r_arr)) {
        this.addResponseCall("tutor.appendSystemText", "OK, that looks like Isabelle is
            happy you've proved the statement. Well done.");
      }
      return this.getResponseStrings();
    } catch (IOException e) {
      String msg = String.format("An exception occurred checking the proof.%n Student
          %s Collection %s Document %s ActionKey %s", this.getUserName(), this.
          getSubCollection(), this.getDocumentId(), this.getActionKey());
      logger.log(Level.SEVERE, msg, e);
      e.printStackTrace();
      throw new ServletException(msg, e);
   } catch (QuestionScriptException e) {
      String msg = String.format("An exception occurred checking the proof.%n Student
           %s Collection %s Document %s ActionKey %s", this.getUserName(), this.
           getSubCollection(), this.getDocumentId(), this.getActionKey());
      logger.log(Level.SEVERE, msg, e);
      e.printStackTrace();
      throw new ServletException(msg, e);
   }
}

/*
 * Checks if Isabelle/HOL thinks the theorem has been proved
 */
protected boolean checkDone(ProverResponseItem[] priArr) {
   for (ProverResponseItem pri: priArr) {
      if (pri.responseLabel == "info" && pri.responseText.startsWith("<html>theorem
           answer:")) {
         return true;
      }
   }
   return false;
}

/*
 * Checking a proof takes approximately 2 seconds. This is too slow to do every
     time the user changes anything (the "changes" we are sent for this question
     are low−level syntax moves), so for this kind of question we do nothing by
     default, and let the user click "Check Proof" to have his/her proof checked.
 */
public void preChangeSetup() throws ServletException, IOException { }
public void preChangeRules() throws ServletException, IOException { }
public void postChangeSetup() throws ServletException, IOException { }
public void postChangeRules() throws ServletException, IOException { }
public void cleanUp() throws ServletException, IOException { }

/**********************************
 * Default advice functions
 **********************************/

public boolean relevance_FailedToFinishProof(String errorCode, String xpath,
     String text) {
   return "Failed to finish proof".equals(errorCode);
}
```

```
public void suggestFix_FailedToFinishProof(String errorCode, String xpath, String
    text) throws ServletException {
  DocumentKey annotationDK = new DocumentKey(this.getUserName(), this.
      getSubCollection(), "annotations_" + this.getDocumentId());
  Document annotationDoc = documentManager.getDocument(annotationDK);
  Document theDoc = documentManager.getDocument(this.getUserName(), this.
      getSubCollection(), this.getDocumentId());

  if (xpathHandler.evaluateToBoolean("count(//mt:tile[@name='answer']/mt:socket/mt
      :tile[@definition='proofs:simp']) > 0", theDoc.getDocumentElement())) {
    this.addCodedResponseCall("recommendFix", "unfinish proof (top level simp)", "
        tutor.prompt", "You can't expect the simplifier to automatically do the entire proof
        for you!");
    return;
  }

  String missingGoal = findUnshownGoal(xpath, annotationDoc);
  if (missingGoal != null) {
    this.addCodedResponseCall("recommendFix", "unfinish proof (found missing goal)
        ", "tutor.prompt", "You still need to show the goal " + missingGoal);
  } else {
    this.addCodedResponseCall("recommendFix", "unfinish proof (can't find missing
        goal)", "tutor.prompt", "Check back in the proof to see if there are any goals that
        you haven't shown");
  }
}

public boolean relevance_UnexpectedEndOfInput(String errorCode, String xpath,
    String text) {
  return "Inner syntax error: unexpected end of input".equals(errorCode);
}
public void suggestFix_UnexpectedEndOfInput(String errorCode, String xpath,
    String text) throws ServletException {
  NodeList nl = xpathHandler.evaluateToList("//mt:tile[@name='answer']//mt:tile/mt:
      socket[not(*)]", documentManager.getDocument(this.getUserName(), this.
      getSubCollection(), this.getDocumentId()));
  if (nl.getLength() > 0) {
    this.addCodedResponseCall("recommendFix", "unfilled sockets", "tutor.highlight"
        , "//mt:tile[@name='answer']//mt:tile/mt:socket[not(*)]", "0xFFAAAA");
    this.addCodedResponseCall("recommendFix", "unfilled sockets", "tutor.prompt",
        "It looks like this is being caused because you haven't filled in some sockets in some
        earlier tiles.");
  } else {
    this.addCodedResponseCall("recommendFix", "incomplete wrong tile", "tutor.
        prompt", "I think you've put something that isn't an expression (maybe a rule label
        ) in an expression socket, but I'm just guessing.");
  }
}

public boolean relevance_TileInIllegalLocation(String errorCode, String xpath,
    String text) {
  return ("Opening PGIP tag found when state is writingPGIP".equals(errorCode) ||
      "Output found outside of a command".equals(errorCode) || "Illegal application of
      proof command in prove mode".equals(errorCode));
}
public void suggestFix_TileInIllegalLocation(String errorCode, String xpath, String
    text) throws ServletException {
  this.addCodedResponseCall("recommendFix", "tile in illegal location", "tutor.
      prompt", "This usually means the tile with the error is in an illegal location (eg an
      equation in a 'proof method' socket).");
}

public boolean relevance_LocalStatementWillFailToSolveAnyPendingGoal(String
    errorCode, String xpath, String text) {
  return ("Local statement will fail to solve any pending goal".equals(errorCode));
}
public void suggestFix_CannotRewriteStatement(String errorCode, String xpath,
    String text) throws ServletException {
  this.addCodedResponseCall("recommendFix", "cannot rewrite statement", "tutor.
      prompt", "If the statement doesnt have a \"For All\" in it, and doesn't have a
      declared external variable, Isabelle won't know what variable she can rewrite");
}

public boolean relevance_CannotRewriteStatement(String errorCode, String xpath,
    String text) {
  return ("Cannot rewrite statement".equals(errorCode));
}
public void suggestFix_LocalStatementWillFailToSolveAnyPendingGoal(String
    errorCode, String xpath, String text) throws ServletException {
  this.addCodedResponseCall("recommendFix", "local statement will fail to solve any
      pending goal", "tutor.appendSystemText", "Here's a topic link for advice on solving
      this one: " + this.topicRecommendLink("solving isar goals", null, "solving isar
      goals", true));
}

public boolean relevance_ProofCommandFailed(String errorCode, String xpath,
    String text) {
  return "empty result sequence -- proof command failed".equals(errorCode);
}
public void suggestFix_ProofCommandFailed(String errorCode, String xpath,
    String text) throws ServletException {
```

```
DocumentKey annotationDK = new DocumentKey(this.getUserName(), this.
    getSubCollection(), "annotations_" + this.getDocumentId());
Document annotationDoc = documentManager.getDocument(annotationDK);

Element e = findClosestPremiseState(xpath, annotationDoc);
if (e != null) {
  List<SimpleNode> premiseList = new ArrayList<SimpleNode>();
  List<SimpleNode> goalList = new ArrayList<SimpleNode>();
  Set<String> varNames = null;

  NodeList nl = e.getChildNodes();
  for (int i = 0; i < nl.getLength(); i++) {
    Node n = nl.item(i);
    if (n instanceof Element && n.getLocalName().equals("premise")) {
      isar isar = new isar(new StringReader(n.getTextContent()));
      try {
        SimpleNode expr = isar.Expression();
        premiseList.addAll(expr.getConstraints(false));
        if (varNames == null) {
          varNames = expr.getIdentifiers();
        } else {
          varNames.addAll(expr.getIdentifiers());
        }
      } catch (ParseException e1) {
        this.addCodedResponseCall("recommendFix", "proof command failed (can't
            parse premise)", "tutor.prompt", "Sorry, I'd try to check if the expression
            was true, but I'm having trouble parsing this premise: <br />" + n.
            getTextContent());
        return;
      } catch (TokenMgrError e1) {
        this.addCodedResponseCall("recommendFix", "proof command failed (can't
            parse premise)", "tutor.prompt", "Sorry, I'd try to check if the expression
            was true, but I'm having trouble parsing this premise: <br />" + n.
            getTextContent());
        return;
      }
    } else if (n instanceof Element && n.getLocalName().equals("goal")) {
      isar isar = new isar(new StringReader(n.getTextContent()));
      SimpleNode goal;
      try {
        goal = isar.Expression();
      } catch (ParseException e1) {
        this.addCodedResponseCall("recommendFix", "proof command failed (can't
            parse goal)", "tutor.prompt", "Sorry, I'd try to check if the expression was
            true, but I'm having trouble parsing this goal: <br />" + n.getTextContent
            ());
        return;
    } catch (TokenMgrError e1) {
        this.addCodedResponseCall("recommendFix", "proof command failed (can't
            parse goal)", "tutor.prompt", "Sorry, I'd try to check if the expression was
            true, but I'm having trouble parsing this goal: <br />" + n.getTextContent
            ());
        return;
    }
    goalList.add(goal);
    premiseList.addAll(goal.getConstraints(true));
    if (varNames == null) {
        varNames = goal.getIdentifiers();
    } else {
        varNames.addAll(goal.getIdentifiers());
    }

    HashMap<String, Object> varsMap = new HashMap<String, Object>();
    try {
        if (findCounterExamples(varsMap, varNames, premiseList, goalList)) {
            StringBuilder sb = new StringBuilder("This counter−example shows the
                line is wrong:<br />");
            for (String varName : varNames) {
                sb.append(varName);
                sb.append("=");
                sb.append(varsMap.get(varName));
                sb.append(" ");
            }

            this.addCodedResponseCall("recommendFix", "proof command failed (
                found counterexample)", "tutor.prompt", sb.toString());
            return;
        } else {
            this.addCodedResponseCall("recommendFix", "proof command failed (can
                't find counterexample)", "tutor.prompt", "I can't find a counter−
                example. Perhaps the line is true but Isabelle can't prove it − maybe you
                need to use an extra rule, or it might just be algebraicly too far from the
                previous line");
            return;
        }
    } catch (EvalException e1) {
        this.addCodedResponseCall("recommendFix", "proof command failed (eval
            failed)", "tutor.prompt", "Sorry, I'd try to check if the expression was true,
            but I had trouble evaluating one of the goals: <br />" + e1.getMessage());
        return;
    }
}
```

```
        }
        this.addCodedResponseCall("recommendFix", "unfinish proof (found missing goal)
            ", "tutor.prompt", documentManager.getXmlContents(e));
    } else {
        this.addCodedResponseCall("recommendFix", "unfinish proof (can't find missing
            goal)", "tutor.prompt", "Didn't find any states with premises " + xpath);
    }
}

/*********************************
 * Functions useful for checking proofs
 *********************************/

/**
 * Tries to find goals that Isabelle hasn't declared shown
 */
protected String findUnshownGoal(String xpath, Document annotationDoc) {
    String missingGoal = null;
    if (annotationDoc != null) {
        String searchPath = "//state[@xpath=\"" + xpath + "\"]/xml/goal";

        NodeList goals = xpathHandler.evaluateToList(searchPath, annotationDoc.
            getDocumentElement());
        NodeList infos = xpathHandler.evaluateToList("//info/text", annotationDoc.
            getDocumentElement());
        if (goals != null) {
            for (int i = 0; i < goals.getLength(); i++) {
                String goalText = goals.item(i).getTextContent();
                boolean found = false;
                if (infos != null) {
                    for (int j = 0; j < infos.getLength(); j++) {
                        Node n = infos.item(j);
                        String s = n.getTextContent();
                        if (s.contains("Successful attempt to solve goal by exported rule") && s.
                            contains(goalText)) {
                            found = true;
                            break;
                        }
                    }
                }
                if (!found) {
                    missingGoal = goalText;
                    break;
                }
            }
        }
```

```
  }
  return missingGoal;
}

/**
 * Searches for a state element that contains at least one premise element, and
     has an xpath that is the closest parent of the given xpath
 */
protected Element findClosestPremiseState(String xpath, Document annotationDoc
    ) {
  Element e = null;
  while (e == null && !Util.empty(xpath)) {
    NodeList nl = xpathHandler.evaluateToList(String.format("//state[@xpath=\"%s
        \" and count(xml/premise) > 0]/xml", xpath), annotationDoc.
        getDocumentElement());
    if (nl.getLength() > 0) {
      e = (Element)nl.item(0);
    } else {
      int i = xpath.lastIndexOf('/');
      if (i > 0) {
        xpath = xpath.substring(0, i);
      } else {
        xpath = null;
      }
    }
  }
  return e;
}

/*
 * Find a counterexample, and put it in varMap; return true if a counterexample has
     been found
 */
protected boolean findCounterExamples(Map<String, Object> varMap, Set<String
    > varNames, List<SimpleNode> premises, List<SimpleNode> goals) throws
    EvalException {
  String[] varNamesArr = varNames.toArray(new String[0]);
  if (varNamesArr.length == 0) {
    return false;
  }
  return doTrials(varMap, 0, varNamesArr, premises, goals);
}

/*
 * Tries different numbers for each of the variables to find a counterexample
 */
```

```
protected boolean doTrials(Map<String, Object>m, int i, String[] varNames, List<
    SimpleNode> premises, List<SimpleNode> goals) throws EvalException {
  for (int k = 0; k < 20; k++) {
    m.put(varNames[i], k);
    if (i + 1 < varNames.length) {
      boolean b = doTrials(m, i + 1, varNames, premises, goals);
      if (b) {
        return true;
      }
    } else {
      StringBuilder sb = new StringBuilder();
      for (int localI = 0; localI < varNames.length; localI++) {
        sb.append(varNames[localI]);
        sb.append('=');
        sb.append(m.get(varNames[localI]));
        sb.append(' ');
      }
      boolean passesPremises = true;
      for (SimpleNode n : premises) {
        Object o = n.eval(m, this);
        if (o instanceof Boolean && !Boolean.valueOf((Boolean)o)) {
          passesPremises = false;
          break;
        }

      }
      if (passesPremises) {
        for (SimpleNode n : goals) {
          Object o = n.eval(m, this);
          if (o instanceof Boolean && !Boolean.valueOf((Boolean)o)) {
            return true;
          }
        }
      }
    }
  }
  return false;
}

public Object call(Map<String, Object> variableValues, FunctionCallback
    functionCallback, String functionName, SimpleNode... parameters) throws
    EvalException {
  throw new EvalException("Couldn't obtain an executable definition of function " +
      functionName);
}
}
```

APPENDIX B

Teaching Script for a Formal Proof Exercise

This appendix contains the Teaching Script for a question about the Fibonacci sequence and the Greatest Common Denominator. It specifies which question document and conversion script to use, defines some *ad hoc* advice functions, and provides an executable definition for the $fib(n)$ and $gcd(n, m)$ functions.

```
package cam.cl.intelligentBook.discreteMaths.questions.gcd;

import java.io.IOException;
import java.util.Map;
import java.util.Vector;
import javax.servlet.ServletException;
import cam.cl.intelligentBook.isabelleExpr.EvalException;
import cam.cl.intelligentBook.isabelleExpr.FunctionCallback;
import cam.cl.intelligentBook.isabelleExpr.SimpleNode;
import cam.cl.intelligentBook.proof.ProofQuestionScript;
import cam.cl.intelligentBook.questions.QuestionScriptException;

/**
 * Teaching script for Question A
 */
public class Scripta extends ProofQuestionScript {

  public Scripta() throws ServletException, QuestionScriptException {
    super();
    this.contextXPath = "/mt:document/mt:tileSet[@name='question']/";

    this.adviceMap = new String[][] {
        {"advice_ruleTiles", "returnTrue", "help"},
        {"advice_fibDef", "returnTrue", "help"},
    };
```

```
}

public String getDocumentId() {
  return "question.xml";
}

public String getSubCollection() {
  return "questions/gcd/a/";
}

@Override
public Vector checkProof() throws ServletException {
  /*
   * We use a Conversion Script that includes the Fibonacci sequence definition
   */
  return checkProof(Scripta.class.getResource("FibonacciIsar.groovy").getPath());
}

public boolean returnTrue() {
  return true;
}

public void advice_ruleTiles() throws ServletException {
  addCodedResponseCall("suggest", "rulesTiles", "tutor.prompt", "Isabelle's simplifier
      only knows a few rules; there are rule tiles in the tray to add more rules. This means
      Isabelle DOESN'T know those rules unless you tell her about them!");
}

public void advice_fibDef() throws ServletException {
  addCodedResponseCall("suggest", "fibDef", "tutor.prompt", "We want to show
      something about gcd( f(n+1), f(n+2) ), and we know that f(n+2) = f(n) + f(n+1) ...");
}

@Override
public Object call(Map<String, Object> variableValues, FunctionCallback
    functionCallback, String functionName, SimpleNode... parameters) throws
    EvalException {
  if ("f".equals(functionName)) {
    if (parameters == null | parameters.length < 1) {
      throw new EvalException("Call to f had fewer than one parameter");
    }
    return fib(getInt(parameters[0].eval(variableValues, functionCallback)));
  } else if ("gcd".equals(functionName)) {
    if (parameters == null | parameters.length < 2) {
      throw new EvalException("Call to gcd had fewer than two parameters");
    }
```

```
            return gcd(getInt(parameters[0].eval(variableValues, functionCallback)), getInt(
                  parameters[1].eval(variableValues, functionCallback)));
        } else {
            return super.call(variableValues, functionCallback, functionName, parameters)
                  ;
        }
    }

    public static int fib(int i) {
        if (i > 0) {
            double sqrt5 = Math.sqrt(5.0);
            double a = Math.pow((1.0 + sqrt5) / 2.0, i);
            double b = Math.pow((1.0 − sqrt5) / 2.0, i);
            double c = (a − b)/sqrt5;
            return (int)Math.rint(c);
        } else {
            return 0;
        }
    }
    public static int gcd(int x, int y) {
        if (x == 0 && y ==0) {
            return 0;
        } else {
            int n = Math.max(x, y);
            int d = Math.min(x, y);
            if (d == 0) {
                return n;
            } else {
                int r = n % d;
                return (r == 0) ? d : gcd(r, d);
            }
        }
    }
    static int getInt(Object o) throws EvalException {
        if (o instanceof Integer) {
            return ((Integer) o).intValue();
        } else if (o instanceof Long) {
            return ((Long) o).intValue();
        } else if (o instanceof Short) {
            return ((Short) o).intValue();
        } else {
            throw new EvalException(String.format("Couldn't get an integer from %s %s", o.
                  getClass().getName(), o.toString()));
        }
    }
}
```

APPENDIX C

Conversion Script for Fibonacci Sequence

The Conversion Script shown here defines the Fibonacci sequence in Isabelle/HOL, and provides matchers for tiles associated with it. It loads the `proofs.groovy` and `maths.groovy` Conversion Scripts to handle all other tiles.

```
String MATHSTILES_NS = "http://www.cl.cam.ac.uk/users/whb21/MathsTiles";

// Matchers for various rule labels
proc.matcher(MATHSTILES_NS, "tile", { it.getAttribute("definition") == "fibonacci:fibm
      " }, {
  out.append("fibm");
});
proc.matcher(MATHSTILES_NS, "tile", { it.getAttribute("definition") == "fibonacci:fib0
      " }, {
  out.append("fib0");
});
proc.matcher(MATHSTILES_NS, "tile", { it.getAttribute("definition") == "fibonacci:fib1
      " }, {
  out.append("fib1");
});
proc.matcher(MATHSTILES_NS, "tile", { it.getAttribute("definition") == "fibonacci:
      fib_add" }, {
  out.append("fib_add");
});
proc.matcher(MATHSTILES_NS, "tile", { it.getAttribute("definition") == "fibonacci:
      gcd_fib_Suc_eq_1" }, {
  out.append("gcd_fib_Suc_eq_1");
});
proc.matcher(MATHSTILES_NS, "tile", { it.getAttribute("definition") == "fibonacci:
      gcd_add2" }, {
  out.append("gcd_add2");
```

```
});
proc.matcher(MATHSTILES_NS, "tile", { it.getAttribute("definition") == "fibonacci:
      gcd_commute" }, {
   out.append("gcd_commute");
});
proc.matcher(MATHSTILES_NS, "tile", { it.getAttribute("definition") == "fibonacci:
      gcd_non_0" }, {
   out.append("gcd_non_0");
});
proc.matcher(MATHSTILES_NS, "tile", { it.getAttribute("definition") == "fibonacci:
      gcd_mult_add" }, {
   out.append("gcd_mult_add");
});
proc.matcher(MATHSTILES_NS, "tile", { it.getAttribute("definition") == "fibonacci:
      gcd_mult_cancel" }, {
   out.append("gcd_mult_cancel");
});
proc.matcher(MATHSTILES_NS, "tile", { it.getAttribute("definition") == "fibonacci:
      gcd_commute" }, {
   out.append("gcd_commute");
});

// Matcher for the induction tile that uses the definition of the Fibonacci series for its
      cases
proc.matcher(MATHSTILES_NS, "tile", { it.getAttribute("definition") == "fibonacci:
      inductionFib" }, {
   out.append("<proofstep>proof (induct ");
   proc.process(it, "mt:socket[@name='variable']");
   out.append(" rule: fib_induct)</proofstep>");

   proc.talk(it);

   proc.process(it, "mt:socketList[@name='step list']");
   proc.process(it, "mt:socket[@name='show']");

   out.append("<proofstep>qed</proofstep>");
   proc.talk(it);
});

// Load the default conversion scripts that contain definitions for various maths and
      proof tiles
proc.use("proofToIsar.groovy");
proc.use("mathsToIsar.groovy");

// Matcher for the document as a whole (always processed first)
proc.matcher(null, null, null, {
   out.append("<aborttheory/>");

   //Set up the alternative induction rule
   out.append("""
   <opentheory>theory question imports Main Primes begin</opentheory>
   <theoryitem>
theorem altInduct [case_names base step]: "P 0 ==> (!!n::nat. P n ==> P (n + 1)) ==> P n"
by(auto elim!: nat_induct)
   </theoryitem>

   <theoryitem>
theorem altCases: "[| m = 0 ==> P; m = k + 1 ==> P |] ==> P" sorry
   </theoryitem>
""");

   proc.talk();

   // Define the Fibonacci sequence and associated lemmas
   out.append("""
   <theoryitem>
consts f :: "nat => nat"
recdef f less_than
"f 0 = 0"
"f (Suc 0) = 1"
"f (Suc (Suc x)) = f x + f (Suc x)"
   </theoryitem>

   <theoryitem>
lemma fib0: "f 0 = 0" by simp
lemma fib1: "f 1 = 1" by simp
lemma fibSuc: "f (Suc (Suc n)) = f (Suc n) + f n" by simp
lemma fibm: "m > 0 ==> f (m + 1) = f m + f (m - 1)" by (cases m, auto)

declare fib0[simp]
declare fib1[simp]
declare fibSuc[simp]
declare fibm[simp]

declare fib1[simplified, simp]
          lemma [simp]: "0 &lt; f (Suc n)"
                by (induct n rule: f.induct) (simp+)

              theorem fib_induct:
                  "P 0 ==> P 1 ==> (!!n. P (n + 1) ==> P n ==> P (n + 2)) ==> P (n::nat
                    )"
                by (induct rule: f.induct, simp+)
```

```
    </theoryitem>

    <theoryitem>
theorem gcd_fib_Suc_eq_1: "gcd (f n, f (n + 1)) = 1" sorry
    </theoryitem>
    <theoryitem>
theorem fib_add: "f (n + k + 1) = f (k + 1) * f (n + 1) + f k * f n" sorry
    </theoryitem>
    <theoryitem>
theorem gcd_mult_add: "0 &lt; n ==> gcd (n * k + m, n) = gcd (m, n)" sorry
    </theoryitem>
""");

    proc.talk();

    // Process the question tile containing the theorem and socket for the proof
    proc.process(it, "//mt:tile[@name='answer']");
});
```

APPENDIX D

Question Document for a Formal Proof Exercise

The question document for a formal proof exercise contains the tiles that will be converted to declare and prove the theorem in Isabelle/HOL. (The tiles that declare the theorem are in the document at the start; the student adds the tiles to prove the theorem.) Question documents are not usually handwritten, but created by piecing the tiles together in MathsTiles and then marking some of them as unselectable and indelible. However, the question document for a question about the Fibonacci sequence is shown here.

```
<document
xmlns="http://www.cl.cam.ac.uk/users/whb21/MathsTiles"
xmlns:d="http://www.cl.cam.ac.uk/users/whb21/DOMEditors"
xmlns:mt="http://www.cl.cam.ac.uk/users/whb21/MathsTiles"
name="question">
<d:requires name="proofs" uri="proofs.xml"/>
<d:requires name="maths" uri="maths.xml"/>

<tileSet name="question" xmlns="http://www.cl.cam.ac.uk/users/whb21/MathsTiles">

<tile definition="proofs:theorem with (is ) slot" name="answer" x="0" y="0" selectable=
    "no" delible="no" background="0xBBEEAA">
<socket name="theorem">
<tile definition="maths:=" name="t6" x="153" y="20">
<socket name="var1">
<function name="gcd" separator="," socketCount="2">
<socket name="var1">
<function name="f" separator="," socketCount="1">
<socket name="var1">
<variable name="n" x="215" y="330"/>
</socket>
</function>
</socket>
```

```
<socket name="var2">
<function name="f" separator="," socketCount="1">
<socket name="var1">
<tile definition="maths:+" name="t5" x="267" y="338">
<socket name="var1">
<variable name="n" x="374" y="257"/>
</socket>
<socket name="var2">
<variable name="1" x="394" y="257"/>
</socket>
</tile>
</socket>
</function>
</socket>
</function>
</socket>
<socket name="var2">
<variable name="1" x="257" y="332"/>
</socket>
</tile>
</socket>
<socket name="is slot">
<tile definition="proofs:(is )" name="t20" x="183" y="149">
<socket name="is1">
<tile definition="proofs:P()" name="t21" x="117" y="207">
<socket name="var1">
<variable name="n" x="138" y="224"/>
</socket>
</tile>
</socket>
</tile>
</socket>
<socket name="proof" selectable="yes" delible="yes" background="no"/>
</tile>
</tileSet>
</document>
```