An Intelligent Tutoring System for Induction Proofs

An Intelligent Tutoring System for Induction Proofs

Abridged Project Description

A. Bundy, J.D. Moore, C. Zinn

Division of Informatics, University of Edinburgh

1 Introduction

There is an increasing use of formal methods in the conceptualisation of the design of safety and security critical hardware (e.g., microprocessor design) and software systems (e.g., authentication protocols and scheduling algorithms). The complexity of such systems calls for computer-assisted development and analysis using formal specifications. Verifying that a given formal specification has desirable properties requires sophisticated proof machinery. Because of theoretical limitations–undecidability results, incompleteness of inductive inference and combinatorial explosion–we cannot aspire to fully automate the theorem proving process. Therefore, proving non-trivial properties of non-trivial specifications requires interaction between a human and the proof machine. The human guiding the machine through an enormous search space; the machine assisting the human by carrying out routine proof tasks and bookkeeping activities. To be effective, each partner must cooperatively communicate with the other in order to complement their expertise (to get the best possible "synergy effect"). From this perspective, a good proof environment is characterised by the quality of the interaction that can be established between the human and the machine.

Taking the increasing importance of formal proofs into account, there is a general need to teach formal methods. We claim that new and better ways to teach and learn formal methods have to be explored. Taking into account the fact that current state-of-the-art proof machines require human interaction, there is a particular need to create proof development environments that allow human and machine to cooperate effectively. This implies that the human user must complement his expertise in formal methods, to a considerable extent, with knowledge of how a given proof system performs proofs and where and how he can make his problem solving contributions.

Goal. This project aims to improve the accessibility and usability of computer assisted proof systems. By building a tutoring system for the OYSTER/CLAM induction prover, we will both explore ways of improving human-computer interaction in proof development systems, and provide a tool for teaching formal proofs. The prospective tutoring system, tentatively called INTUITION, will therefore specialise in induction proofs, and will assist students of mathematics and informatics, as well as formal methods practitioners to learn and to perform these kinds of proof using OYSTER/CLAM.

Research questions. Due to the intrinsic complexity of formal proof, creating an environment for teaching formal proof is by no means trivial. A fundamental problem for traditional teaching in this domain is that the "learning by doing" paradigm can be carried through for only very small and unrealistic problems [BE]. For explaining and solving interesting problems, blackboard, chalk, paper and pencil alone are not sufficient, and therefore, a new tool, computer-assisted proof construction, has to be developed. Building such a computer-supported learning environment for induction proofs has to cope with a wide range of problems, all closely interconnected:

Q1 The induction proof domain can be characterised by the absence of a well-established *common* stock of expert knowledge for solving induction problems. Heuristic knowledge is either taught in the classroom by the teacher, or acquired by the student through practice performing induction proofs. The textbook literature on "how to solve induction problems" is sparse, but see [Sol90,Vel94]. *Can we collect existing expert knowledge on induction proving and encode it?*

Q2 Formal proofs require a formal language. Learning such a new and artificial language and using it correctly, is the first hurdle that students must overcome. The language used in textbook proofs is far more a natural than a formal language, but much less concise. *What languages are effective for teaching induction theorem proving?*

Q3 The formal proofs generated by most current proof machines exhibit an unacceptable level of detail. These proofs are tediously long and therefore it is hard, if not impossible, to understand them. In comparison, proofs found in textbooks are written at a much higher level of abstraction. Therefore, they are much shorter and, in principle, much more readable. However, textbook proofs do not meet the criteria of formal correctness. *How can we simultaneously achieve understandability and formal correctness?*

Q4 Contemporary proof machines have poor explanation capabilities. If they fail to generate a proof for a given conjecture, then they cannot explain their failure. If they succeed, then they can explain neither their proof success nor the proof itself. *How can we generate helpful explanations for proofs and proof failures?*

Q5 No two students are equal. A good student will use more advanced methods to construct a proof than a weak student. Consequently, a weak student needs a different kind of assistance than a strong student. The frequency of tutor-student interaction, the pacing, the kind of explanations or hints should reflect this fact. *How can we deliver personalised teaching in this domain?*

In this project we will test the hypothesis that proof plans are an enabling technology for INTUITION's design. In particular, we wish to demonstrate that proof plans: (i) facilitate the mathematical understanding of a proof; (ii) allow the teaching of the mathematical concepts of proof construction; (iii) provide the framework for supporting natural interaction between tutor and student; and (iv) act as the underlying representation needed to support the generation of high-level proof explanations.

2 The Programme

2.1 Addressing the Research Issues Raised in Questions Q1-Q5

A1 As part of developing the OYSTER/CLAM induction prover, *a large corpus of expert knowledge has been acquired and encoded.* In fact, Edinburgh possesses the largest collection of expert knowledge in the induction domain. In particular, to direct search, a goal-directed rewriting strategy, called "rippling" [BSvH+93], has been automated. In addition, heuristics for lemma speculation and generalisation have been developed and implemented. There is also a large collection of teaching experience and instructional material available.

A2 There is evidence that different presentations of the same material affect the ways that people with different cognitive styles can be taught. Therefore, there will be no single best language to teach induction proof, and therefore, we will pursue *a multi-modal approach.* We intend to supplement a graphical proof presentation/construction with a natural language presentation/construction of the proof. Of course, parsing natural language is difficult because one has to cope with a wide range of linguistic phenomena. However, the given context facilitates this task: the domain of discourse is restricted; the tutor-student communication is goal-oriented; and proof plans provide structures which lead to strong expectations about possible discourse continuations [Zin99]. Also, it will only be necessary to provide parsing for a fragment of English, namely the concise subset of English which is used in textbook proofs and which we want the student to learn.

We hypothesise that proof plans will facilitate natural interaction between tutor and student. In particular, they provide a rich representation of abstract proof strategies that is necessary for sophisticated natural language understanding and generation in this domain.

A3 Proof plans are an attempt to capture the common structure of proofs, and are used to guide the proof search [Bun88]. Proof planning has been implemented in the OYSTER/CLAM system [BvHHS90]. CLAM builds a proof plan customised to the current conjecture and then uses this plan to instruct the theorem prover, OYSTER, how to prove the theorem. All the search is conducted during proof planning; the execution of the proof plan requires no search. During proof search, instead of carrying out elementary steps at the inference level, *tactics* are applied. A tactic is a program that describes the application of a sequence of inference rules. Tactics are formally described by *methods* which specify the preconditions that must hold for a tactic to be carried out. The OYSTER/CLAM system implements an AI plan formation algorithm which seeks to construct a proof plan by searching for a tree of methods that combine to prove a given conjecture. *The proof planning technology allows the generation of high-level proofs. These proof plans lead to formal proofs if each method can successfully execute its associated tactic.*

A4 *Proof plans are an enabling technology for generating high-level proof explanations.* Since proof plans capture the common structure of proof, they facilitate the mathematical understanding of a proof. From the learning per-

spective, proof plans allow the teaching of the mathematical concepts of proof plan construction — in contrast to formal proof construction in, say sequent calculus. Also, proof plans distinguish between interesting (outside a proof plan) and routine (inside a proof plan) proof steps. In addition, OYSTER/CLAM, unlike other proof machines, is also able to report on the cause of its proof failure by using its proof critic and patch facility [IB94]. *Critics* can be attached to methods and are used to analyse their failure. A critic looks at the preconditions and the proof context, and can then suggest a *proof patch* by supplying additional information to the prover (e.g., an intermediate lemma). Proof plans also provide a better basis for user interaction since the state of the proof can be described and controlled at a higher level than is usually possible. This is a crucial issue for proofs which cannot be achieved automatically.

A5 *Effective explanation is essentially incremental and interactive*, that is, more coaching-like and less lecturing-like. Moore's prior work shows that explanations should take the current problem solving context, the student's knowledge and prior discourse into account [Moo93,Moo96]. Therefore dynamic generation of explanatory text from underlying knowledge structures is required. Proof plans provide the abstraction necessary to recognise when a similar strategy is used to solve different problems and when mistakes are being made in similar contexts. We plan to exploit this representation to enable our explanation generator to point out similarities between problem solving situations, relate new material to old material, and provide alternative explanations to repeated errors.

2.2 XBARNACLE *as a Starting Point.*

Figure 1 depicts a screen shot of XBARNACLE [LD97,Jac99], a semi-automated version of OYSTER/CLAM. A (partial) proof is represented as a graph where each node is labelled with a proof obligation. At the beginning of proof construction, there is only the root node carrying the conjecture. The user can press two "forward" buttons to develop the proof. Pressing the "simple forward" button will generate the next proof step, pressing the "fast forward" button will generate as many proof steps as possible. Each node carries a status indicating whether the sub-proof that originates at this node is open (not yet developed), partially developed, or complete. If the underlying proof system is unable to develop a node, this nodes gets the status "stuck". Some nodes represent sub-plans and a more detailed proof can be requested by unpacking these nodes. For each node, its applicable methods as well as any associated proof critics can be viewed. If OYSTER/CLAM cannot prove a given conjecture automatically, the user is requested to interact with the proof critics and proof patch facility. XBARNACLE's explanatory capability is based on a canned text approach. XBARNACLE is complemented with tools that support browsing through a list of currently loaded theorems, definitions, wave rules and reduction rules. — XBARNACLE *is therefore a good starting point for the prospective tutoring system.*

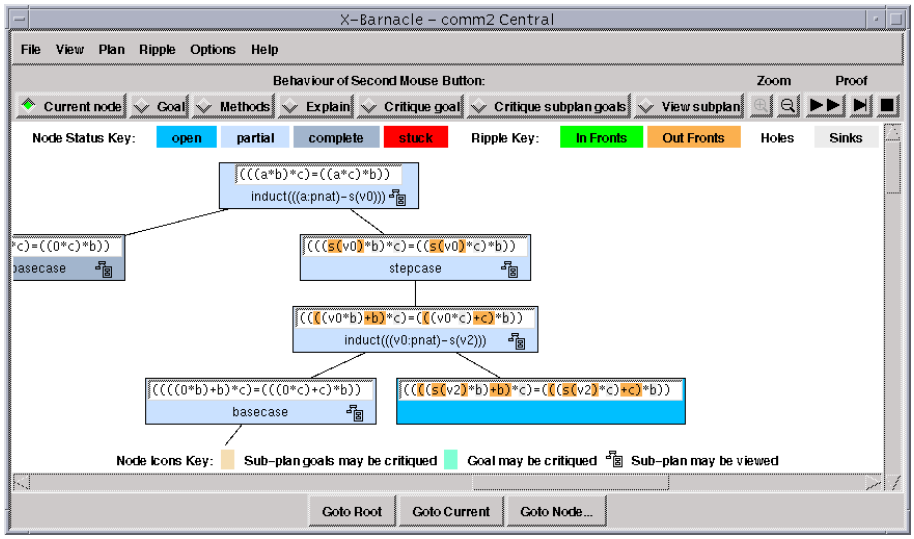

Fig. 1. XBarnacle in Action.

2.3 Architecture of the Tutoring System.

The instructional goals of the tutoring system will be: (i) to teach students how to perform induction proofs ("learning by doing") by adopting the proof planning paradigm, and by applying domain dependent heuristics to direct the proof search; (ii) to teach students how Oyster/Clam performs induction proofs, its strengths and limits, and how they can contribute to the proof construction. The tutoring system is designed to complement, not to replace classroom teaching. Tutoring will be more coaching-like than lecturing-like. That is, instead of re-lecturing the instructional material that has been presented in classroom instruction, the tutor will help only when asked, and only in the form of hints encouraging the student to consider this or that possibility.

The architecture we envision for Intuition is depicted in Fig. 2. The major part of the expert module is Oyster/Clam which encodes most of the knowledge that we currently have for doing induction proofs. The expert module is complemented by a collection of buggy knowledge representing common misconceptions when doing induction proofs. The current problem state is captured explicitly in a data structure external to Oyster/Clam which makes it possible to represent buggy problem states. The instructional module consists of a representation of expert knowledge for induction proofs from the teaching perspective. The student diagnosis and therapy module observes the students' problem solving to derive their performance, and to uncover their strengths and weaknesses. It interfaces with the action interpreter, consults the current problem state and maintains the student model. To repair the student's misconception, or to supply missing knowledge, the tutor generates hints and explanations in natural

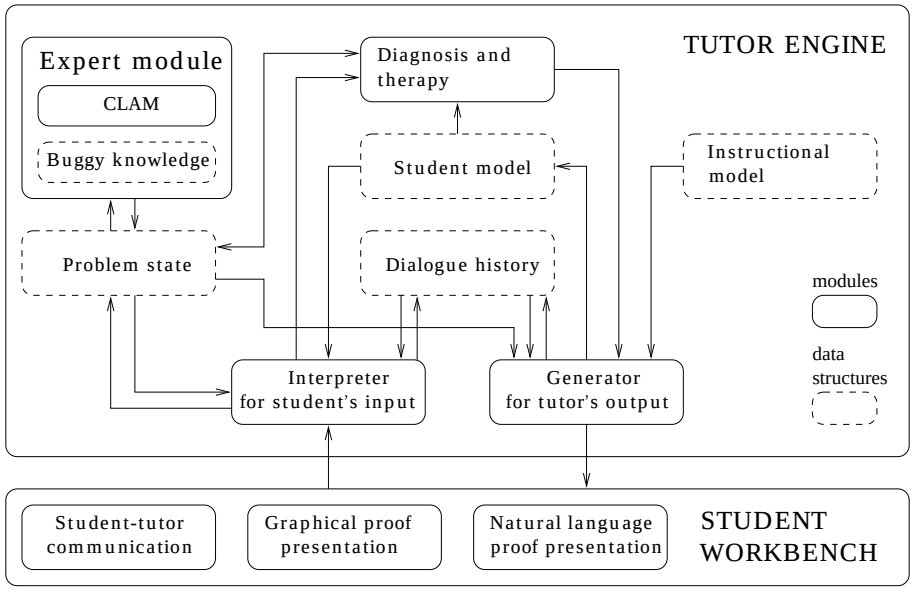

Fig. 2. The Architecture of INTUITION.

language. Explanations take the current problem solving context, the students' knowledge and prior discourse into account. The student workbench provides for a multi-modal tutor-student interaction. The graphical representation of proof plans is supplemented by a natural language representation, both allowing the manipulation of proof plans at different levels of detail.

Our proposed tutoring system, INTUITION, will be built on an existing problem solver, OYSTER/CLAM, representing state-of-the-art induction proving. However, OYSTER/CLAM cannot be a perfect expert and will, in general, need human assistance to prove a conjecture. Therefore, INTUITION must be, in nature, a collaborative system. This adds an additional challenge to the design of intelligent tutoring systems. We must determine how to give feedback when the student requires help on a problem that OYSTER/CLAM cannot solve itself, and we must provide a way for the student to provide proof guidance to OYSTER/CLAM in those situations where the student has more expertise than OYSTER/CLAM. We believe that proof plans provide leverage for solving both of these problems.

Proof plans capture strategic problem solving behaviour that mathematicians use when constructing proofs. Recognising the proof plan facilitates the mathematical understanding of a proof, and thus we hypothesise that teaching students how to recognise and construct proof plans will facilitate the teaching of the mathematical concepts/skills of proof construction. However, in the induction domain there can be many correct proof plans for a given conjecture. It is therefore hard to determine what proof strategy the student is pursuing,

and where along the solution path the student got stuck. INTUITION's diagnosis module will build on proof planning technology in order to cope with this.

3 Related Work

The number of existing tutoring systems is large, and is accompanied by a vast amount of literature. We will only discuss a selection of tutoring systems for the mathematics and logics domain.

Tarski's World [BE92] and Hyperproof [BEA94] are two of the best-known systems for teaching logic. Tarski's World introduces students to the syntax and semantics of first-order logic (FOL) by providing a graphical representation for each FOL sentence and vice versa. Hyperproof introduces students to formal reasoning and proof construction. It extends Tarski's World in a natural way by providing heterogeneous inference rules which allow the integration of sentential and graphical representations of FOL. An evaluation of Hyperproof provides evidence (i) that the system can stand comparison with well-established teaching methods, and (ii) that students should be encouraged to broaden their representational repertoire (because being able to interleave visual and verbal reasoning is an advantage) [SCO95]. Assuming that efficient reasoning is heterogeneous in nature, future versions of Hyperproof will allow additional types of diagrammatic reasoning providing, e.g., Euler circles, Venn diagrams, and logic matrices.

Jape is an interactive tool for teaching and doing formal reasoning [SB96]. It provides for different logics (e.g., sequent calculus, set theory, equational reasoning), a logic encoding mechanism for describing new logics, two graphical proof representations (tree-like, box-like), direct manipulation of the proof, and a tactics mechanism. Contrary to other interactive theorem provers where the user's intelligence is only required when the theorem prover is stuck, Jape is designed to be a proof environment where the human has to direct proof construction from the beginning.

MathPert is a learning tool for mathematics supplying teaching modules for algebra and calculus [Bee98]. It is able to analyse any standard problem in these domains, and supplies not only a solution but also a correct series of steps leading to it. MathPert is easy to use and can assist students in solving problems by offering hints and proposing intermediate steps in response to a student's request for help. However, there is currently no sufficient evaluation of MathPert's impact on teaching.

None of the above systems provides individualised teaching or feedback (which necessitates a student and diagnosis module) and therefore, they should not be called intelligent tutoring systems. In contrast, the systems developed by Anderson and colleagues (the Lisp [AR85], algebra [RACM98] and geometry [KA93] tutors) maintain a student model to determine what feedback to give and what problems the student should be presented with. These "cognitive tutors" are based on Anderson's ACT theory of learning which claims that knowledge can only be learned by doing. The principal step for building a tutor is to develop a cognitive model of expert and student problem solving in a particular domain by

analysing expert problem-solving strategies as well as common student misconceptions. The cognitive model is then used for "model tracing", a technique for recognising the student's individual plan, as correct or erroneous, in order to provide assistance or encouragement. Model tracing also allows the system to trace the students' knowledge development across a range of skills to be acquired, and allows for individual pacing by selecting appropriate problems. The mathematics tutors, which aim to support students and teachers, have been marketed, are in widespread use, and have been proven effective in several large-scale evaluations [ACKP95].

4 Conclusion

Using automated theorem provers, like OYSTER/CLAM, it is now possible to formally verify industrially significant properties of software and hardware. However, the user of the prover must be highly skilled and the development time is of the order of man-months — sometimes man-years. Such skill levels and development times preclude widespread industrial use. This has restricted the use of formal methods to niche markets where the need for reliability outweighs the high costs and the delays.

There is ongoing research to enhance the practicability of theorem provers by reducing both the skill levels and the development time required to verify systems. A major factor in doing this is to automate the proof process as much as possible and to make interaction easier, when automatic proof is not sufficient. The UITP ("User Interfaces for Theorem Provers") workshop series aims to disseminate design principles of user interfaces for theorem proving assistants [AGMT95,SB96]. We argue, however, that having good interfaces is not enough. Because a complete automation of proof is still not possible, it is also necessary to improve the expertise of the human user, who plays the crucial part in the development of proofs. Therefore, it is timely to develop a computer-supported teaching tool that supports users in acquiring a high level of skill which will enable them to effectively cooperate with theorem provers.

The immediate beneficiaries of our work will be the users of OYSTER/CLAM: students that need to acquire formal methods, and formal methods practitioners. They will have a tool that provides individualised teaching, and which should significantly reduce the time for learning and applying formal methods. Teachers may find INTUITION a useful complement to classroom teaching, and a source of inspiration to adapt their teaching. More widely, we anticipate that the tutoring technology we have developed will then be taken up by the developers of other theorem proving systems and bring them similar benefits. In particular, it should make such tools commercially more attractive. Also, our approach may contribute to the design of intelligent tutoring systems.

References

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

[AGMT95] J. S. Aitken, P. Gray, T. Melham, and M. Thomas. Interactive proof discovery: An empirical study of HOL users. In P. Gray, editor, *User Interfaces Design for Theorem Proving Systems*, pages 1–8, 1995. International Workshop: Glasgow.

[AR85] J. R. Anderson and B. J. Reiser. The Lisp tutor. *Byte*, 10(4):159–175, 1985.

[BE] J. Barwise and J. Etchemendy. Computers, visualization and the nature of reasoning. Available from: `http://www-csli.stanford.edu/hp/Logic-software.html`.

[BE92] J. Barwise and J. Etchemendy. *The language of first-order logic*. Stanford, Calif. CSLI, 1992.

[BEA94] J. Barwise, J. Etchemendy, and G. Allwein. *Hyperproof*. Stanford, Calif. CSLI, 1994.

[Bee98] M. Beeson. Design principles of Mathpert: Software to support education in algebra and calculus. In N. Kajler, editor, *Computer-human interaction in symbolic computation*. Springer-Verlag, 1998.

[BSvH+93] A. Bundy, A. Stevens, F. van Harmelen, A. Ireland, and A. Smaill. Rippling: A heuristic for guiding inductive proofs. *Artificial Intelligence*, 62:185–253, 1993.

[Bun88] A. Bundy. The use of explicit proof plans to guide inductive proofs. In R. Lusk and R. Overbeek, editors, *Ninth Conference on Automated Deduction*, pages 111 – 120. Springer Verlag, 1988.

[BvHHS90] A. Bundy, F. van Harmelen, C. Horn, and A. Smaill. The oyster-clam system. In M. E. Stickel, editor, *Proceedings of the 10th International Conference on Automated Deduction*. Springer-Verlag, 1990.

[GCV98] A.S. Gertner, C. Conate, and J. VanLehn. Procedural help in Andes: Generating hints using a bayesian network student model. In *American Association for Artificial Intelligence*, 1998.

[IB94] A. Ireland and A. Bundy. Productive use of failure in inductive proof. *Journal of Automated Reasoning*, 16, 1994.

[Jac99] M. J. Jackson. *Interacting with semi-automated theorem provers via interactive proof critics*. PhD thesis, School of Computing, Faculty of Engineering and Computing, Napier University, 1999.

[KA93] K.R. Koedinger and J.R. Anderson. Reifying implicit planning in geometry: Guidelines for model-based intelligent tutoring system design. In S. Lajoie and S. Derry, editors, *Computers as Cognitive Tools*. Hillsdale, NJ: Erlbaum, 1993.

[LD97] H. Lowe and D. Duncan. XBarnacle: Making theorem provers more accessible. In *CADE-14*, 1997.

[MLR96] J. D. Moore, B. Lemaire, and J. A. Rosenblum. Discourse generation for instructional applications: Identifying and exploiting relevant prior explanations. *Journal of the Learning Sciences*, 5(1):49–94, 1996.

[Moo93] J. D. Moore. What makes human explanations effective? In *Proceedings of the Fifteenth Annual Meeting of the Cognitive Science Society*. Lawrence Erlbaum Associates, 1993.

[Moo96] J. D. Moore. Making computer tutors more like humans. *Artificial Intelligence in Education*, 7(2):181–214, 1996.

[RACM98] S. Ritter, J. Anderson, M. Cytrynowicz, and O. Medvedeva. Authoring content in the PAT algebra tutor. *Journal of Interactive Media in Education*, 1998.

[SB96] B. Sufrin and R. Bornat. User Interfaces for Generic Proof Assistants – Part I: Interpreting Gestures. In *User interfaces for Theorem Provers, York, U.K.*, 1996. Part II: Displaying Proofs is available at `http://www.comlab.ox.ac.uk/oucl/users/bernard.sufrin/jape.html`.

[SCO95] K. Stenning, R. Cox, and J. Oberlander. Contrasting the cognitive effects of graphical and sentential logic teaching: reasoning, representation and individual differences. *Language and Cognitive Processes*, 10:333–354, 1995.

[Sel99] J. Self. Open Sesame? : fifteen variations on the theme of openness in learning environments. *International Journal of Artificial Intelligence in Education*, 10:1020–1029, 1999.

[Sim90] D. L. Simon. *Checking Number Theory Proofs in Natural Language*. PhD thesis, UT Austin, 1990.

[Sol90] D. Solow. *How to read and do proofs*. John Wiley & Sons, 1990.

[Vel94] D. J. Velleman. *How to prove it*. Cambridge Univ. Press, 1994.

[Zin99] C. Zinn. Understanding Mathematical Discourse. In *Proceedings Amstelogue'99, Workshop on the Semantics and Pragmatics of Dialogue*. Amsterdam University, 7-9 May 1999.