A Tutorial Program for Propositional Logic with Human/Computer Interactive Learning

Stacy Lukins, Alan Levicki, and Jennifer Burg
Department of Computer Science
Wake Forest University
Winston-Salem, NC 27109
burg@cs.wfu.edu

Abstract

This paper describes a tutorial program that serves a double role as an educational tool and a research environment. First, it introduces students to fundamental concepts of propositional logic and gives them practice with theorem proving. Secondly, the program provides an environment in which we can track student learning, explore cognitive issues of human problem solving, and investigate the possibilities of interactive human/machine learning. We have tested the tutorial program on two groups of Discrete Mathematics students and report the results of our assessment. We also discuss the contributions and future directions of our research in interactive human/machine learning.

1 Background

Propositional logic is a staple of introductory computer science courses, often taught in CS I, Discrete Mathematics, or both. Mastery of the concepts of formal logic, so basic to the problem-solving inherent in computer science, requires practice, particularly in the deductive methods of formal theorem-proving.

To supplement the logic exercises given in typical textbooks, we have created a logic tutorial program called P-Logic Tutor. In an interactive Web-based environment, the program introduces students to concepts of propositional logic and gives them practice in creating well-formed formulas, applying inference rules, and proving theorems.

SIGCSE'02, February 27- March 3, 2002, Covington, Kentucky, USA.

In addition to its usefulness as a teaching tool, the tutorial program serves as a research testbed in which we can monitor student learning patterns and investigate how humans and machines can learn interactively – the human being prompted with clues for how to proceed with a proof, and the computer improving in its ability to provide useful clues over multiple tutorial sessions with multiple students.

This interactive human/machine learning component distinguishes P-Logic Tutor from previous tutorial theorem-proving systems.

Earlier systems have generally fallen into five categories: interactive theorem provers, proof checkers, proof assistants, proof editors, and formal logic tutorial programs. Only the last category puts the emphasis on pedagogy, and none of these has an interactive human/machine learning component.

A number of interactive theorem provers have been created, but they differ from P-Logic Tutor in that they are targeted toward having the computer generate proofs rather than teaching humans how to do so. Their interactivity is motivated by a need to improve the efficiency of fully-automated theorem provers – i.e., those in which the computer does all the thinking. Examples of interactive theorem provers include Nqthm (also known as the Boyer-Moore Theorem Prover) [5, 15, 16], ACL2 [11], HOL [9], and INKA [10]. In these systems, the user can help guide the theorem prover by suggesting lemmas or undoing previous steps. In this sense, such systems do the reverse of P-Logic Tutor, where the computer suggests steps to the user.

Proof checkers, proof assistants, and proof editors put more emphasis on teaching the user how to do a proof, although that is not necessarily their primary purpose. Proof checkers simply determine if the construction of a proof is correct (e.g., Wai Wong et al.'s Proof Checker for HOL

[9].) They are often part of a larger theorem-proving system.

Proof assistants also can help the user to check proofs for correctness, but they provide a more fully-developed environment for constructing proofs. Some include built-in hints for particular problems, often predefined either by the programmer for sample problems, or by the user for user-defined problems. Examples of proof assistants include ETPS (the interactive component of TPS) [18], Coq [7], CPT I (the Carnegie Mellon Proof Tutor project) [6], Kumo [13], and WinKe [20]. All of these assistants provide proof checking. Their user interfaces are quite varied, ranging from command-driven systems (ETPS and Coq) to those with GUIs (WinKE, CPT I, Kumo, and an updated version of Coq called Pcoq [17]).

Proof editors allow users to construct their own proofs in a specified logic. They seek to make the tasks involved in constructing proofs easier by, for example, providing tools for developing proof trees or giving simple methods for applying inference rules to axioms. Those with more sophisticated user interfaces allow the user to view and edit proofs graphically. Proof editors may or may not have the ability to check for the correct application of inference rules or the correct construction of the entire proof. Examples of proof editors include Alfa [8], where proofs are done in natural deduction style; Jape [4], a generic proof editor where the logic formalism can be described in a metalanguage; and the editing component of PLAGIATOR [12].

Tutorial systems that, like P-Logic Tutor, are designed to teach concepts of formal logic include the commercially available packages of Barwise and Etchemendy. Tarski's World introduces students to the language of first-order logic through reasoning with graphical objects [2]. Its extension, Hyperproof, is a system for learning the principles of analytical reasoning and proof construction [3]. The Logic Tutor of Abraham et al. [1] also helps computer science students learn techniques of formal proofs, but it is distinguished by its adaptability to different logic languages (i.e., prepositional and predicate) and by its ability to characterize types of mistakes and respond to them appropriately. The main emphasis in these systems is not on generating proofs for some other application, but on teaching humans how to reason in logic formalisms.

The word "intelligent" is sometimes applied to tutorial systems. Generally, intelligent tutoring systems are those that do not simply give generic feedback to student mistakes. Instead, they tailor the feedback to the type of mistake, and some also develop a student profile over time so that they can individualize problems and feedback accordingly. (See [19] for a review of intelligent tutoring systems.)

We use the term "intelligent" in a different way as applied to P-Logic. P-Logic Tutor is an intelligent tutorial system in that it has its own built-in theorem-proving ability, and it is designed to learn better theorem-proving strategies over time so that it can offer increasingly useful clues. However, P-Logic Tutor is also extensible to intelligence in the more conventional sense in that student activity is monitored and stored and can be evaluated in subsequent tutorial sessions for customized feedback.

2 P-Logic Tutor: A Tutorial Program Combining Pedagogy and Research

The primary purpose of P-Logic Tutor is to teach students fundamental concepts of propositional logic and theorem-proving techniques. The tutorial is divided into three learning modules:

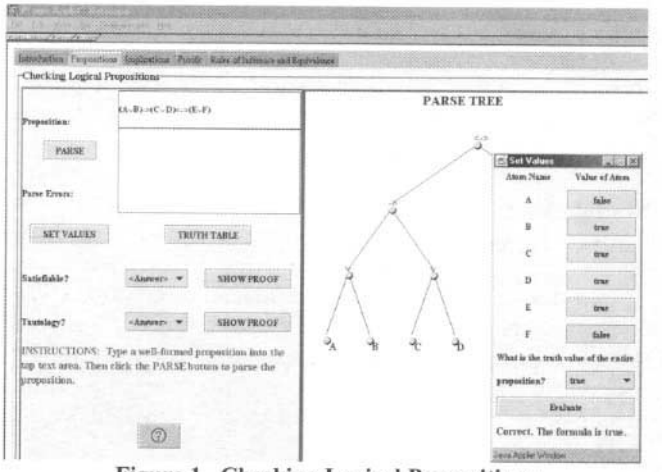

Figure 1. Checking Logical Propositions

Module 1 – In this module, the student can

- attempt to compose a well-formed formula and check its syntax;
- ask that the formula be parsed;
- pop open a truth table;
- enter values in individual rows and columns of the truth table and check their correctness;
- ask that the truth table be automatically filled in;
- set truth values for atomic propositions within the formula, say what the resulting truth value for the formula would be, and check correctness;
- say if the formula is satisfiable and check correctness; and
- say if the formula is a tautology and check correctness.

Figure 1 gives a view of Module 1.

Module 2 – In this module, the student can

- attempt to show how a conclusion can be derived from one or two axioms (using problems that are automatically generated by the system);

- tell what rule of inference or equivalence justifies this derivation;
- check for correctness; and
- see the proper rule application if his/her answer is incorrect.

Figure 2 gives a view of Module 2.

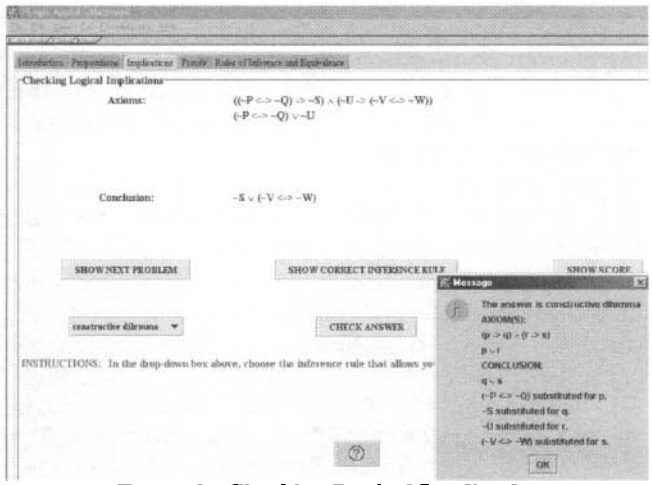

Figure 2. Checking Logical Implications

Module 3 – In this module, the student can

- attempt to prove a theorem by incrementally applying rules of inference and equivalence (using problems that are automatically generated by the system);
- input a problem of his/her own, consisting of axioms and a theorem to prove; and
- request a "clue" from the tutorial program as a proof proceeds.
-

Figure 3 gives a view of Module 3.

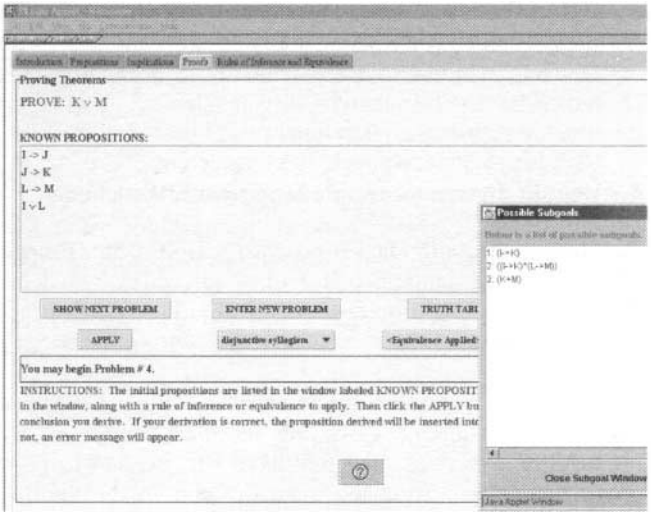

Figure 3. Proofs

At any time during a tutorial session, the student can pop open a Help window, which gives help both on the user interface and the subject matter itself, i.e., concepts in propositional logic.
An additional tabbed pane is always available containing a list of rules of inference and equivalence applicable to the problems and proofs of Modules 2 and 3.

P-Logic Tutor serves as a learning tool in a second way, providing a research vehicle through which we can examine how students solve logic problems and how humans and machines might be helped to learn from each other. This aspect of P-Logic Tutor exists behind the scenes of the tutorial system and fuels the investigations of the faculty/student team that has worked on the project in the past two years.

3 The Implementation

3.1 Implementation of the Tutorial System

P-Logic Tutor is implemented in Java and accessible from the Web at www.cs.wfu.edu/~burg/JavaPackages/indexswingnet.html. Visitors can log in with the user name *anonymous* and password *plogic*.

The purpose of requiring student log-ins is to identify the student at each tutorial session so that his/her activity can be monitored and saved at the tutorial Web site. These files provide us with our assessment data, and they can also be used for later customization of a user's difficulty level and feedback.

The program also includes a context-sensitive Help feature implemented by means of the JavaHelp extension. Students can pop up the Help window at any time during a tutorial session to look up definitions of terms in formal logic. Use of this Help window is also monitored, transparent to the user.

3.2 Implementation of Interactive Theorem-Proving

The tutorial system described above is fully implemented and has been tested on two groups of students. In the meantime, we are designing and have partially implemented an interactive learning system to enhance the effectiveness of the tutorial environment.

The learning system is divided into four components divided into two phases, as pictured in Figure 4.

The *theorem prover* consists of the environment in which the user can construct proofs, including facilities for allowing the user to apply inference and equivalence rules and checking the correctness of their application. In addition, the theorem prover can present potentially useful steps, or subgoals, to the student. This component is fully operational and has been tested on student users.

The remaining three components are still in the design and implementation stage, but since they operate behind the scenes as enhancements to the tutorial process, we are able to use the tutorial system with students while we continue to develop these components.

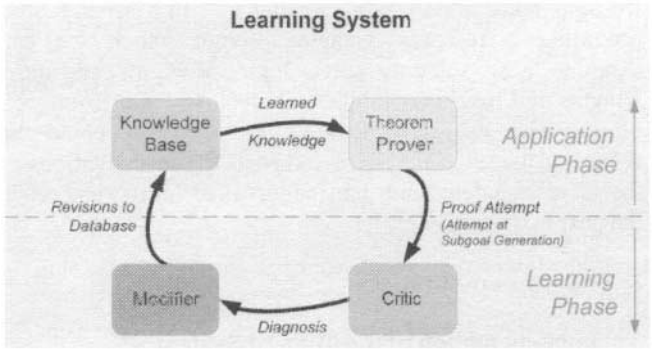

Figure 4. A Model of the Learning System

The purpose of the *knowledge base* is to store data about past proof problems, proof types, and methods that work well with certain problem types. This is the tutorial system's memory.

The purpose of the *critic* is to analyze what takes place in the theorem-proving component. The critic determines whether a suggested subgoal is ever used in the construction of a proof. It also analyzes whether a suggested subgoal contributes to an efficient proof. (Measures of an "efficient" proof include the number of steps taken in the proof compared to other known solutions, and the number of subgoals generated that are never ultimately used in the proof.) The results of the critic's analysis are sent to the *modifier*.

The purpose of the *modifier* is to determine when revisions should be made to the knowledge base. The modifier can store new data, modify existing data, or name and store new inference rules.

3.3. Suggesting Subgoals

We have spent the majority of our time thus far on the subgoal-suggestion feature within the learning system.

Subgoals are suggested by four methods:

(1) Parse trees of the axioms are created. Shared atomic propositions are constrained to have the same values. A recursive procedure propagates logic through the parse tree for each axiom, and where the truth values of atomic variables or subexpressions can be determined, these are put on a list of provable subgoals.

(2) Expressions are transformed into equivalent forms. Equivalence rules that are particularly useful include *distribution*, *absorption*, and *DeMorgan*'s laws. It is the role of the *critic* to identify types of problems to which these rules best apply. For now, we check their applicability in all problems by searching for expressions that match their pattern. When an expression has been transformed to an equivalent form, it can be suggested as a subgoal or processed through one of the other methods in this list.

(3) Logic is propagated across expressions that share operands. For example, if we know that $P \rightarrow Q$ is true and $P \wedge Q$ is false, then P must be false, so $\neg P$ can be suggested as a subgoal. For another example, consider a problem with the axioms $Y \rightarrow Z$, $Z \rightarrow [Y \rightarrow (R \vee S)]$, $R \leftrightarrow S$, and $\neg(R \wedge S)$. The expressions $R \leftrightarrow S$, $R \wedge S$, and $R \vee S$ share the variables R and S. If the truth values of the first two are known – true and false, respectively – then they necessitate that the third be false. Thus, $\neg(R \vee S)$ could be suggested as a subgoal.

(4) Some atomic variable P is set to either *true* or *false*, and its logical conclusion, if there is one, is determined. If some subexpression *S* must be true where P is true, then $P \rightarrow S$ may be suggested as a subgoal.

4 Assessment of the Tutorial System

4.1 Implementation Issues

Although Java has the advantage of being Web-portable, its frequently changing versions created some implementation problems. The GUI requires Java's Swing classes, which in turn require the Java Plug-In for Web browsers, and we found that students with an older version of Java and the plug-in on their computers could not access the tutorial. This was quickly remedied with an upgrade. Some classes also did not prove to be as portable as we would have liked. The HTMLEditor class was particularly problematic in the Unix environment, not displaying formatted html properly in the Help window. Aside from these problems, Java proved to be a good language for development and Web-distribution of the tutorial system.

4.2 User Friendliness and Pedagogical Effectivenss

We tested P-Logic Tutor on two classes of Discrete Mathematics students in consecutive semesters. Students reported very few problems in the tutorial system other than some small interface bugs that were uncovered.

Most students found the first two modules quite easy to use. The Proofs module gave students the most trouble mostly because the exercises that they were being asked to do – prove theorems – were the most difficult in the tutorial. Some comments related to user-friendliness. For example, a few students said they would like the panel containing the Rules and Inference and Equivalence to be constantly in view while they are doing a proof. Several said that they

would like a "back" button that would allow them to return to previous problems.

Quite a few comments had to do with the manner in which students were required to prove the theorems. Rules have to be applied in exactly the form they are given, a restriction that some students found unnecessary. For example, before the *conjunctive simplification* rule can be applied to A∧B in order to derive B, A∧B must be commuted to B∧A, since the rule is stored in the form $p \wedge q \ \therefore p$. Allowing implicit commutation would not be difficult to implement, but deciding how many rules we could allow to the student to collapse into one step is trickier with regard to implementation.

The subgoal suggestion feature was considered helpful by many students, but some said that they could have used more guidance in which subgoal to try. (In some cases, other subgoals would have to be proven before the one suggested could be proven. Our intention is to have the computer "learn" the intermediate steps.)

The most interesting comments from students related to their manner of thinking through proofs, which did not always match the rule-application proof method they were asked to put into practice. Many felt that they knew, logically, why a theorem was true, but they couldn't find rules to match the way they were thinking. For example, some wanted to use a substitution method – proving the truth value of a subexpression, substituting it out with *true* or *false*, and using a rule like "*false* ∧ X is always false" to simplify the expression. Alternatively, they may have wanted to do case-base reasoning or proof by contradiction. Applying rules in a syntactical term-rewriting process did not necessarily match their logical thinking process.

5 Future Work

The long-term goals of this project take two branches:

(1) Continue to develop the interactive human/machine learning component of the tutorial system. Investigate and compare theorem-proving methods that are natural to humans and computers, respectively, and determine ways in which each can learn from the other.
(2) Continue to investigate how humans best learn logical thinking. Implement an alternative tutorial module that is more graphical and intuitive (as opposed to a rule-based term-rewriting process), allowing the student to visualize logic propagation and proof strategies. Do pre- and post-testing of students using English-language-based logic problems to determine the extent to which the study of formal logic can improve a person's problem solving ability with real-world logic problems expressed in everyday language.

References

[1] Abraham, D., Crawford, L., Lesta, L., Merceron, A., and Yacef, K. The Logic Tutor: A Multimedia Presentation. *The IMEJ of Computer Enhanced Learning* 3 (2), Oct. 2001. http://imej.wfu.edu/articles/2001/2/index.asp.
[2] Barwise, J. and Etchemendy, J. Logic Software from CSLI. Sept. 2, 2001. http://www-csli.stanford.edu/hp/Logic-software.html.
[3] Barwise, J. and Etchemendy, J. *Hyperproof.* Stanford, CA: CSLI Publications, 1994.
[4] Bornat, R. and Sufrin, B. Jape – A Framework for Building Interactive Proof Editors. Sept. 2, 2001. http://users.comlab.ox.ac.uk/bernard.sufrin/jape.html.
[5] Boyer, R. S. Nqthm, the Boyer-Moore Prover. Sept. 2, 2001. http:// www.cs.utexas.edu/users/boyer/ftp/nqthm/.
[6] The Carnegie Mellow Proof Tutor Project. Sept. 2, 2001. http://hss.cmu.edu/HTML/departments/philosophy/Proof_Tutor.html.
[7] The Coq Proof Assistant Sept. 2, 2001. http://coq.inria.fr/.
[8] Hallgren, T. The Proof Editor Alfa. Sept. 2, 2001. http://www.cs.chalmers.se/~hallgren/Alfa/.
[9] The HOL Theorem Proving System. Sept. 2, 2001. Laboratory for Applied Logic, Brigham Young University. http://lal.cs.byu.edu/lal/hol-documentation.html.
[10] The Inductive Theorem Prover, INKA, Version 4.0. Sept. 2, 2001. http://www.dfki.de/vse/systems/inka/.
[11] Kaufmann, M. and Moore, J. S. ACL2 Version 2.5. Sept. 2, 2001. http://www.cs.utexas.edu/users/moore/acl2/acl2-doc.html.
[12] Kolbe, T. and Brauburger, J. PLAGIATOR – A Learning Prover. *Proceedings CADE-14.* Springer LNAI 1249, 1997.
[13] Kumo: A Web Based Proof Assistant. Sept. 2, 2001. http://www.cs.ucsd.edu/groups/tatami/kumo.
[14] LogiCoq. Sept. 2, 2001. http://wims.unice.fr/~wims/it_U3~logic~logicoq.en.html.
[15] Moore, J. S. The Boyer-Moore Theorem Prover (NQTHM). Sept. 2, 2001. http://www.cs.utexas.edu/users/moore/best-ideas/nqthm/index.html.
[16] PcNqthm: An Interactive "Proof-checker" Enhancement of the Boyer-Moore Theorem Prover. Sept. 2, 2001. http://www.cli.com/software/pc-nqthm/index.html.
[17] Pcoq: A Graphical User Interface for Coq. Sept. 2, 2001. http://www-sop.inria.fr/lemme/pcoq/.
[18] TPS: Theorem Proving System. Sept. 2, 2001. http://gtps.math.cmu.edu/tps.html.
[19] Urban-Lurain, Mark. Intelligent Tutoring Systems: An Historic Review in the Context of the Development of Artificial Intelligence and Educational Psychology. Sept. 7, 2001. http://aral.cse.msu.edu/Publications/ITS/its.htm.
[20] WinKe: A Proof Assistant for Teaching Logic. Sept. 2, 2001. http://www.dcs.kcl.ac.uk/~endriss/WinKE.