A Computer Tutor for Formal & Applied Logic

A Proposal to NCR Corporation

Preston K. Covey
Associate Professor of Philosophy
Vice Provost for University Studies
Director, Center for Design of Educational Computing

January 8, 1988

Please see Appendix (gray pages at end) for illustrations

Outline of Contents

The Center for Design of Educational Computing (CDEC), in collaboration with faculty in the Department of Philosophy, has made major progress with modest funding this year in the development of a unique intelligent tutor to guide and teach (a) the construction of proofs in mathematical logic and formal theoretical domains (eg., Boolean algebra, social choice theory, elementary probability theory . . .) and (b) logical problem solving (eg., the formal reconstruction of arguments, logic puzzles and word problems presented in natural language).

We have no funding for this project beyond June '88. A grant on the order of $50K would support (full time) one applications programmer and (part time) a systems programmer and research scientist to continue our work, with philosophy faculty, for another year, while we seek follow-on funding. I outline here the significant work we could accomplish even with this level of funding and the very visible impact it would have, especially on campus, in the coming year.

Current Status of the CMU Proof Tutor

Our brochure (Appendix), prepared for the presentation of our work-in-progress at the recent December '87 convention of the American Philosophical Association, describes the functions and ambitions of the CMU Proof Tutor. I further describe below the intellectual agenda and educational significance of our Proof Tutor. To date we have achieved the following unique functionalities:

* **The proof generator.** We have developed a proof generating algorithm in a highly portable, economical but full Common Lisp: (1) that embodies the proof discovery techniques of expert human logicians (an expert system) in propositional logic (with indirect proof), (2) that is demonstrably complete (i.e., has been proven to generate all and only valid proofs -- an important theoretical result), and (3) that has been tested on the standard benchmark problems and scores of other notoriously difficult proof problems resulting in (from the expert and pedagogical viewpoints) elegant and perspicuous proofs.

Items (1) and (3) make our tutor unique with respect to resolution-method automated theorem provers, whose algorithms (while complete) are quite arbitrary from the human expert's strategic viewpoint and quite unhelpful from the novice's viewpoint: such algorithms are inadequate bases for apt or intelligent advice to novices because (a) they operate on arbitrary strategies rather than human-expert heuristics and (b) are not sensitive in any given context to myriad apt proof strategies that might be elected by the user. Item (2) makes our proof generator unique with respect to one other human-heuristic based generator, which works 'most of the time' but is neither formally and demonstrably complete nor any longer in use.

* **The display-based interface.** We have developed (in CDEC's own graphics-oriented programming and authoring language, CMU Tutor) a prototype interface for the Proof Tutor with four key features that, in combination, make our environment absolutely unique: (1) it allows the user either or simultaneously (a) to work within the standard top-down proof format or (b) to work bottom-up by the step-wise subgoaling method logicians actually use and recommend; (2) it represents the tree-structure of the backward subgoaling reasoning perspicuously and graphically; (3) it allows the user to make tentative steps in the subgoal tree, graphically highlighting those steps requiring proof, thus allowing the user to plan and sketch a proof strategy informally, as is common expert practice; and (4) the execution of steps in a proof or help requests can be either keyboard/command driven or entirely mouse driven, as the user prefers (obviating the frustration of typographical errors).

I describe the interface, somewhat redundantly, as ***display-based*** to highlight two crucial features: regarding ***i/o execution***, everything the user needs to know is displayed on the screen and whatever the user needs to do can be done by manipulating what is displayed on the screen with the mouse; regarding ***knowledge representation*** and the intellectual work the tutor is designed to facilitate, the *strategic knowledge* (goal structure) of the requisite reasoning is represented graphically on the screen and the *rule-based knowledge* the user needs at any step (rule structure) is formally displayed in a dialog box (see illustration in Appendix). Thus the interface accomodates all principal styles for proof imaging, planning and i/o execution.

We have yet to implement the Proof Generator's *procedural knowledge* -- about how to indentify appropriate *goals* and proceed forwards or backwards among goals by appropriate *rules* of logic -- as advice specially tailored to an assessment of an individual user's own progress, but generic procedural knowledge is now presented as dialog as illustrated by the simple example in the screen photo in the Appendix. The translation of the Proof Generator's procedural knowledge into advice responsive to individualized diagnoses of users' needs will be a major project over the coming year(s), through formative evaluation. The various help facilities will be controllable with switches that can limit the help available to users for homework or selected problems. As it stands, this interface design is a major advance on any extant proof construction environment.

* **Communication between the proof generator and the display-based user interface.** The Proof Generator (as well as the parsing and proof-checking machinery) is written in a highly portable, public domain Common Lisp. Common Lisp is especially suited to the implementation of these functions, but *not* for the design and implementation of a display-based interface. The latter is therefore implemented in CMU Tutor, also portable across major hardware and operating systems. But the Lisp and CMU Tutor facilities are built to be able to call each other. This has important implications not only for the Proof Tutor itself, but also for its utilization within a larger instructional environment: an on-line, computer-managed course in first-order logic and applied domains -- the VALID program developed at Stanford University.

Proposed Activities

What would be accomplished with NCR support

Of the following activities, an NCR grant would ensure completion of 1 (a) and (b), 2, 3 and 4. There would be high probability of completion of 5, some progress on 6, but, with the projected manpower and the empirical work involved, 7 is an opportunistic wish-list item, as is 1 (c).

The progress assured on items 1 through 4 would nonetheless ensure highly visible deployment in a large and very important course (the only totally on-line course on campus) and *tour de force* presentations at next year's conferences. The resulting system would also be shared and showcased at Stanford University and, possibly, other schools in the InterUniversity Consortium for Educational Computing, for which we are headquarters.

Since I am chair of the American Philosophical Association's Committee on Computer Use in Philosophy, organize its annual software fair and computing conference, and publish the *Computers & Philosophy* quarterly, the results of the proposed work (an operational and deployable version of the proof tutoring environment in an on-line course as well as stand-alone) would be assured effective showcases in the profession and nationally.

1. Upgrading the VALID on-line course & integrating the CMU Proof Tutor.

The on-line, computer-managed course in first-order logic and applied domains (elementary arithmetic, Boolean algebra, elementary probability theory, social choice theory), called VALID, was developed at Stanford University over fifteen years ago, has been deployed there since 1973 and is in use at dozens of other sites around the world, including Carnegie Mellon. Our on-line course enrolls about 100 students a semester, about twice the number that have historically enrolled in our traditionally-taught introductory logic course or the math department's analogue. The VALID-based course is popular with technical students because it is self-paced, very convenient, effective enough, and a novel experience.

The fact is, however, that this program is a dinosaur; its scroll and command driven user interface is atrocious; so much so that I, for one, cannot stand to use it for logic problem solving or theorem proving work. And when I've tutored students from the course, I 've used my own proof-checker on the IBM PC to do VALID's problems. (Today I would use our Proof Tutor environment.) Ideally, an environment such as VALID, apart from its course-support function, should be eminently inviting and usable for doing such work. It is not.

The power of the environment as it stands derives from several factors: something of the kind (a proof-checking environment) is vastly better than nothing (although, except for the error-checking facility, this one is no easier to work with than paper-and-pencil); it allows self-paced progress through its curriculum (which is very extensive but shallow); it contains excellent exercises and problem sets on-line, including the theorem-proving exercises in applied domains (above) to reinforce the transferability and utility of natural deduction skills; and the computer-management of exercises and grading is invaluable to instructors (and would be extremely expensive to replicate for more user-friendly systems, including our Proof Tutor).

Another downside of the program is that it runs in an antiquated Lisp on mainframe computers that CMU (among others) is phasing out.

The basic utility of the VALID program, dated though it is, is nonetheless great, too great to discard. The sensible thing to do would be to upgrade the program where it is weak and preserve its unique values (the course-management facilities and variety of excellent exercises and domain applications). The straightforward upgrade would consist of the following:

a) **Porting/converting the program** from its antiquated Utah Standard Lisp into a public domain Common Lisp, namely the Kyoto Common Lisp (KCL) in which we are building our Proof Tutor. The Stanford team (at the Institute for Mathematical Studies in the Social Sciences, IMSSS) agrees and is eager to cooperate. KCL is written in and compiles to C code and therefore is portable across UNIX environments and modern advanced-function workstations. This portability and the public-domain status of KCL makes the investment worthwhile (and KCL, while slower than other full Common Lisps that have correspondingly larger core images, is an adequate performer). This would allow VALID to run and be supported on the growing base of advanced-function workstations as well as mini's and mainframes. With workstations as cycle servers, the program could also be run from micros on a network. This move would bring VALID along into the 1990's. But, more importantly, this port would enable the two following upgrades:

b) **Integrating the CMU Proof Tutor with the VALID on-line course.** This would be made possible by the port of VALID to KCL on the workstations. Wide deployment for these integrated environments would be enabled by running them from micros off cycle servers over a network (analogous to the current access to VALID on mainframes from micros or

terminals). Large-capacity workstations could support VALID and the Proof Tutor locally.

This integration, supplanting the proof construction machinery of VALID with our Proof Tutor, would have large pay-offs for both systems: (i) the proof-construction and help facilities available to students doing proofs in the on-line course would be vastly improved and (ii) the Proof Tutor would be installed, used and tested in a totally on-line, computer-managed course (it would still, of course, be available as a free-standing environment for deployment in other course settings or for personal use by students and faculty).

c) **Further enhancements to VALID's interface and curriculum.** While this work would not be a priority for next year on the requested funding, the port of VALID to KCL would allow several further types of opportunistic enhancement.

Given that our graphics-oriented programming and authoring language, CMU Tutor, now communicates with KCL functions, other improvements to VALID's basic interface (like interactive on-line help facilities, graphic displays) and VALID's currently shallow tutorial curriculum could be easily and incrementally built up. In fact, development in CMU Tutor is so fast and easy that individual faculty could tailor their own tutorial lessons at will. Nifty graphic displays could be devised with CMU Tutor as illustrations for the Boolean algebra, propositional and predicate logic tutorials (eg., analogue circuit logic, truth-function and Venn diagrams).

Given that KCL can also call C routines, modules like CDEC's Symbolization Tutor (in C, for first-order predicate logic) could be integrated as well (to replace VALID's current stilted translation exercises and machinery). Also, in VALID's social choice theory curriculum, a nice interface could be built to call CDEC's PD World (a graphic simulation environment for N-person iterated Prisoner's Dilemma-type games) in an adjacent window, which exists in both C and CMU Tutor versions.

It makes eminent sense to enhance and *build upon* rather than abandon the intellectually rich VALID environment -- eminently possible once it is ported to KCL. ***With the requested funding, we would be able to port VALID to KCL, integrate our Proof Tutor environment with the on-line course, and deploy the enhanced course and Proof Tutor in the second semester of 1988-89.***

(This in turn would pave the way for similarly upgrading and integrating Stanford's other mighty behemoth environments for set theory and proof theory, building towards a comprehensive ***interactive computer-based 'encyclopaedia'*** for first- and higher-order logics, with associated tutors, on distributed advanced-function workstations -- CDEC's long-term goal.)

2. Fully implementing the display-based interface, with advice & help facilities. With the requested funding, we would be able to make our prototype interface fully operational with help and advice facilities, integral with the Proof Generator, for course deployment stand-alone as well as in the on-line logic course by second semester of 1988-89.

3. Formative evaluation + design of the student model/diagnostician. These parallel activities would take place concommitantly with activities 1 and 2 and continue through AY 1988-89, informing the development of the first-generation help and advice facilities for course deployment second semester of '88-89. A full-blown student model and diagnostician able

to generate very fine-grained advice optimally tailored to the particular local needs of individual students would not be fully implemented before several iterations with several classes had been completed -- a long-term, albeit incremental, project.

4. Installing our translation facilities in the Proof Tutor environment.

CDEC has constructed another intelligent tutor, a Symbolization Tutor called CSYM, for generating and guiding solutions to symbolization problems -- translations between natural language and the artificial language of first-order predicate logic, a major bottleneck in logic curricula. The parsing and generative machinery are already well developed if not fully refined, in C. Because KCL can call C routines, this machinery can and would be integrated in the Proof Tutor environment, to allow the user to get (a) automated English translations or paraphrases of formulae or (b) automated formalizations (complete or partial) of English sentences.

Such a facility for (a) can be very helpful to students inexperienced in the formal language of logic for reasoning through steps of a proof or understanding the logical force of those steps, even in totally abstract proof exercises. But this is especially helpful when the proofs relate to a theoretical domain (like social choice or probability theory) where an intuitive understanding of a proof or theorem's content can aid or help motivate one's reasoning.

Such a facility for (b) is specifically helpful in applied derivation tasks where one must first formalize an argument, a piece of discourse, a puzzle or word problem in order to use the formal apparatus to analyze or solve it. An analogy: imagine the effect on teaching/learning algebra if students could receive tutorial aid in translating the infamous word problems into equations.

The application of our Proof Tutor environment to the analysis or solution of substantive, content-ful problems presented in natural language will be an important one, especially for teaching some formal proof concepts and apparatus to non-technical audiences and (eventually) high school students, where an intuitive understanding of the concepts and techniques -- and what difference they make in applied contexts and substantive problem solving-- is paramount.

5. Extending the proof-generation capability to predicate logic. This activity will proceed apace with the above, and will probably be straightforward, but it is difficult to predict the rate of progress before we are about it. The apparatus for propositional logic with indirect proof (easily extended at minimum to monadic predicate logic) will be plenty powerful for course deployment and support in '88-89.

6. Constructing interactive, on-line tutorials in proof construction. This is easily done incrementally in CMU Tutor. Short of encyclopedic coverage of all the relevant concepts and techniques of natural deduction, prototype interactive tutorials can be devised on the essential concepts, rules, and techniques, with working exampless that can be plugged in for practice in the Proof Tutor's workbench.

7. Constructing a working student model & diagnostician. The refinement of such a facility is a long-term project, requiring close user-testing throughout several iterations of design. During formative evaluation and given long experience with 'buggy' student proof strategems, we will be able to devise adequate advice facilities exploiting the Proof Generator.

Significance, Impact & Visibility of the Proposed Project

Access to the Proof Tutor environment itself would be useful wherever some formal derivational work is presupposed. Such courses are not limited to mathematical logic or discrete mathematics where first-order logic is *taught,* but could be any (eg., in philosophy, linguistics, mathematics, or computer science) where first-order predicate logic is *used or assumed* (such as our series in *logic & computability* or *logic, AI & probability*). The Proof Tutor would be useful to faculty, graduate or undergraduate students for individual work as well. The Proof Tutor provides not just a learning environment, but also ***a widely applicable toolkit, like a spreadsheet.***

Thus, the impact is potentially very widespread ***across the curriculum.*** The most visible impact would be in the introductory on-line logic course, whose popularity is bound to increase. This course would become far more inviting and accessible to non-technical students for one thing.

Additionally, the on-line course *cum* Proof Tutor could provide one solution to a widespread problem noticed by several faculty in a variety of contexts: ***the lack of literacy in the basic concepts and techniques of formal proof or argument*** among technical and non-technical students alike. Even students with extensive and successful course experience in executing the mechanics of proof construction typically do not articulately and reflectively internalize either the concepts or strategies of formal proof and argument construction. Conceptual understanding and transfer of proof strategy typically requires explicit training *in the very process* of practice that only a tutor (human or computer) can provide. Practically, this means reliance on a computer tutor that can be available to prompt and assist a student reflectively (rather than mechanically) through her work whenever she sees fit to do her work.

The self-paced course, taken in whole *or in part* (as a mini or short course) could provide self-study recourse for students wanting to enhance their ***formal-logical literacy.*** This applies to non-technical students in non-formal fields (eg. rhetoric or writing) where formal models of argument provide heuristic constraints for informal modes of argumentation.[1,2,3]

The domain-specific parts of the curriculum, with the help of the Proof Tutor, could provide accessible supplements to ***topical studies*** (eg., Arrow's Paradox) in other courses (eg., economics, social philosophy or social choice theory). The application of formal proof techniques across substantive topics or domains pregnant with intuitive meaning to students would be useful for reinforcing the transferability and applicability of literacy in formal proof structures.

Such ***domain-specific applications*** could be garnered from the VALID curriculum or provided by substantive word problems tailored by different instructors to fit their own course contexts (either on-line, using CMU Tutor, or on paper, to be entered and solved in the Proof Tutor environment by the students -- the Proof Tutor will operate with student-entered problems or stored problem sets entered by an instructor). The Proof Tutor's domains of application are easily extensible and can be tailored by individual faculty in myriad ways.

The bottom-line utility of the Proof Tutor -- as either a widely applicable stand-alone facility or an integral part of the on-line course -- is that it makes an important intellectual toolkit more accessible and intelligible to students on many different levels: first-order logic, natural deduction skills and formal proof techniques are fundamental to a very broad range of intellectual work and theoretical domains. The pity is that the subject is typically taught as a specialized technical topic rather than as a widely applicable organon, because it is not readily accessible or intelligible as such to students. The Proof Tutor will help us redress this problem.

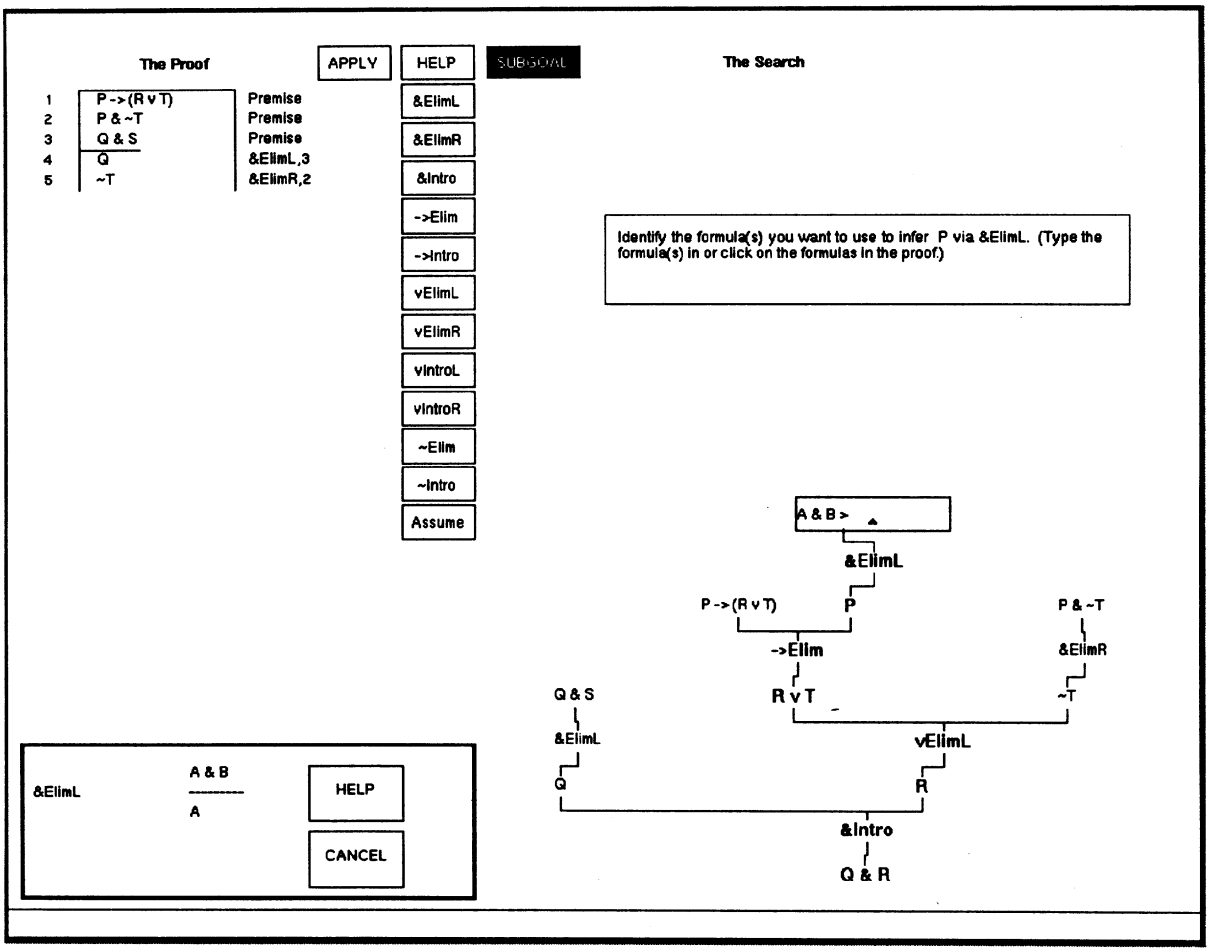
References

1. See Covey "Logic and Liberal Learning" and "Formal Logic and Philosophical Analysis," Beardsley "Logic and Rhetoric," Schwartz "Logic as a Liberal Art," *inter alia* , in Preston K. Covey (Editor) *Formal Logic and the Liberal Arts* , a conference volume published as a special double issue of the journal *Teaching Philosophy,* 4 (3/4) July/October 1981.

2. See also the sections on the applications of computer proof construction environments to the reconstruction of natural language arguments and other modes of philosophic analysis in Preston K. Covey "Computer Assisted Instruction in Philosophy" in Solvieg Olsen (Editor), *Computer Aided Instruction in the Humanities,* New York: Modern Languages Association of America, 1985.

3. For illustration of formal proof concepts applied as heuristic constraints (per the programme in the references above) in teaching writing and rhetoric, see David S. Kaufer and Christine M. Neuwirth "Integrating Formal Logic and the New Rhetoric in Teaching the Argumentative Essay: A Four-Stage Heuristic," *College English* April 1983, and the Comments and Responses on this article in *College English* February 1984.

THE CMU PROOF TUTOR



Jonathan Pressler, Richard Scheines

Chris Walton, and Alan Sobel

Department of Philosophy

and

Center for Design of Educational Computing

CARNEGIE MELLON UNIVERSITY

The Center for Design of Educational Computing at Carnegie Mellon University (CMU) is building an intelligent, computerized tutor that uses its own proof generation expertise to help students to become proficient at constructing natural deduction proofs. Like many other instructional logic programs, our proof tutor will tell students when they have completed a derivation and will give them useful feedback when they enter illegal steps. However, the CMU Proof Tutor will also:

- teach a general, effective **method** for constructing logical derivations;
- provide a **flexible user interface** that exploits diagrammatic displays;
- automatically offer **appropriate strategic advice** at any point in any proof problem in standard logic.

No currently available computerized aid for logic instruction has these features.

In order to provide students with good proof construction advice whenever they need it, a computerized tutor must contain an expert proof generator that knows how to plan and complete any unfinished derivation that a student might produce. Furthermore, the tutor's internal proof generator should search for derivations in much the same way that competent human logicians try to discover them. Otherwise it will be difficult, if not impossible, for the tutor to convert its proof generating expertise into advice that students can use.

Our Proof Generator. We have developed a proof generating algorithm that mimics the proof discovery techniques employed by many human logicians. Roughly speaking, the algorithm searches for a proof by subgoaling backward from the desired conclusion toward the given premises. Although it employs teachable heuristics for deciding which of many subgoals to pursue first, we have proved that our algorithm is a complete decision procedure for propositional logic. It is implemented in Common Lisp on the Andrew computing system at Carnegie Mellon University. The program has been tested on over a hundred difficult proof construction problems, and, in each case, it has produced a reasonably elegant solution in a matter of seconds. This includes all 17 of J. F. Pelletier's benchmark problems for automatic propositional theorem provers.[1] To our knowledge, no other algorithm that has been proven complete for propositional logic models the systematic proof construction techniques used by human experts. During the next three years we will continue to improve our proof generator and extend its domain of expertise to more powerful logics.

The Basic Structure of the Proof Tutor. Conceptually, our proof tutor consists of two parts. One is a set of on-line tutorials that will offer interactive introductions to the concepts and techniques of natural deduction. The other is a proof building environment in which students will receive timely feedback and advice that helps them to develop effective proof generation strategies.

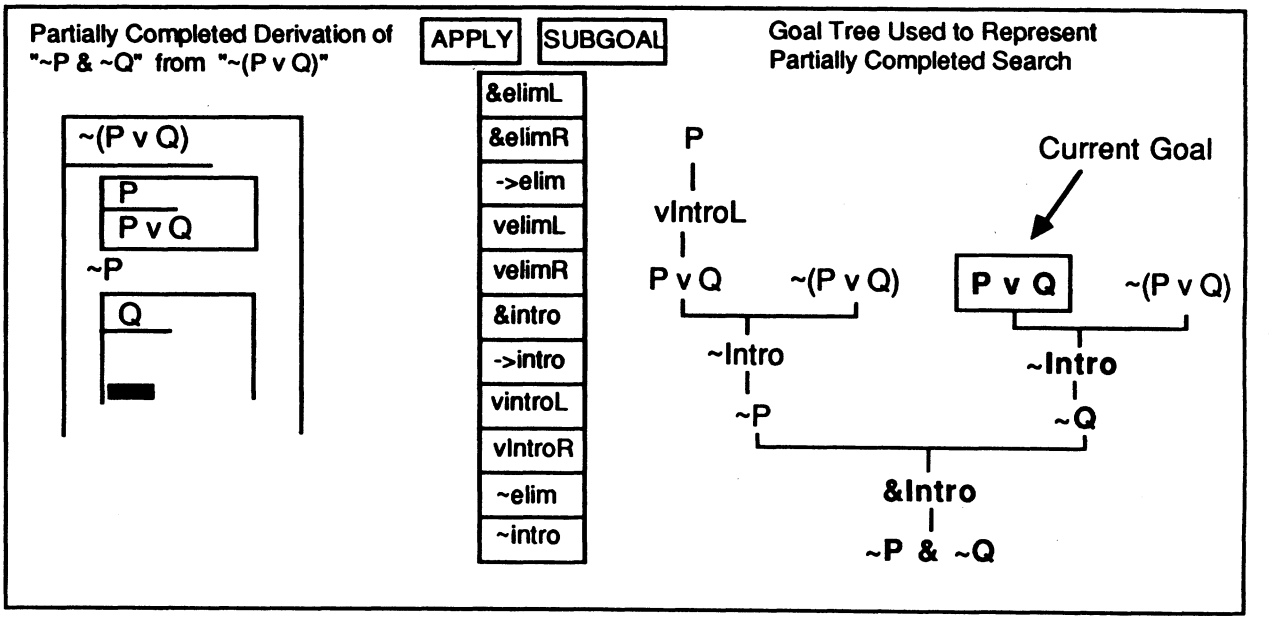
---

[1]Pelletier, J. F. (1986), "Seventy-Five Problems for Testing Automatic Theorem Provers", *Journal of Automated Reasoning* 2, pp. 191-216.

Proof Construction Tutorials. These interactive lessons will teach students how to:

- apply inference rules;
- recognize the logical dependencies that constrain the circumstances under which a line of a proof can be used to justify the addition of new lines;
- formulate candidate subgoals and decide which candidates to pursue first;
- determine when one has achieved a subgoal ;
- keep track of deferred subgoals;
- detect "garden paths" (i.e. search paths that lead either nowhere or to suboptimal proofs);
- recover from strolls down garden paths;
- determine when a proof is complete.

The Proof Building Environment. Students will do proof construction exercises in a "workbench" that is divided into two halves. (See the cover or the figure below.) The left half of a student's workbench will contain a modified Fitch-style representation of his or her *proof* (in its current state of completion); the right half will display a "goal tree" diagram that represents the student's *search* for a complete proof. In the left half of the workbench, students will add steps to a proof by applying inference rules to sentences that are already in the proof. The tutor will check each new step and respond appropriately.



Goal Trees. The goal tree diagram is intended to help students learn how to search for a proof by subgoaling backward from an argument's conclusion toward its premises. The root of a goal tree is the ultimate conclusion that the student is trying to derive. Every other formula in a student's goal tree is a subgoal that is pursued in order to derive that conclusion. Emerging from each goal *g* in the tree is a stem with radiating branches. The stem contains the name of an

inference rule; and each of the radiating branches leads directly to a formula. These formulas are the initial subgoals for generating *g*. *g* can be inferred immediately from the initial subgoals via the rule named on the stem from which those subgoals radiate. Thus, the goal tree in the figure above indicates that "~P" and "~Q" are the initial subgoals for proving "~P & ~Q". It is easy to see that "~P & ~Q" can be inferred immediately from these initial subgoals via the rule &Intro.

In short, goal trees graphically depict when one goal is being used to achieve another. They also employ readily identifiable stylistic conventions that help students to recognize which subgoals are mere candidates, which are actively being pursued, which have been deferred, which have already been attained, and which have already been discovered to be unattainable. Finally, the goal tree keeps the *tutor* apprised of the student's progress in constructing a proof.

In general, whenever a student needs strategic help, the tutor will collect information from both the student's goal tree and the student's partially completed proof diagram. Using these data, the automatic proof generator will internally determine a "best" next step for the student to take. The tutor will then construct a dialogue derived from the process that the generator used to arrive at this step. The dialogue's purpose is to lead the student to discover the very same step by employing the same methods that the tutor's proof generator used. Thus the tutor will explicitly discuss the kind of reasoning needed to conduct efficient searches for elegant proofs.

Communicating with the Tutor. Our tutor already has a congenial user interface that makes it easy to enter information in the workbench described above. Students can "talk" to the tutor by either typing on a keyboard or manipulating a mouse. By using the mouse to choose items on pop-up menus, push on-screen buttons, and select formulas displayed on the screen, students can apply inference rules, introduce subgoals, and transfer fully justified subgoals from goal trees to proofs.

A Stand-alone Tool and a Part of a Comprehensive Logic Teaching Program. The CMU Proof Tutor is intended to function both as a stand-alone tool for teaching natural deduction and as a part of more comprehensive logic teaching programs. VALID, a computerized logic teacher developed at Stanford, will be the first comprehensive logic program to incorporate our tutor. VALID already contains a logic curriculum that covers all of the topics addressed in a standard symbolic logic course. However, there are serious deficiencies in VALID's proof construction environment. Roughly speaking, we will substitute our proof construction environment for VALID's. Since VALID is already being used at many colleges and universities, thousands of students will be able to use our tutor once it is integrated into this comprehensive logic program.

For further information about the CMU Proof Tutor, contact:

Jonathan Pressler
Department of Philosophy
Carnegie Mellon University
Pittsburgh, PA 15213
(412) 268-8566

Richard Scheines
Center for Design of Educational Computing
Carnegie Mellon University
Pittsburgh, PA 15213
(412) 268-8533