

A Computer Environment for Writing Ordinary Mathematical Proofs

David McMath, Marianna Rozenfeld, and Richard Sommer

{mcdave,marianna,sommer}@epgy.stanford.edu
Education Program for Gifted Youth
Stanford University
Stanford, California 94305-4115

Abstract. The EPGY Theorem-Proving Environment is designed to help students write ordinary mathematical proofs. The system, used in a selection of computer-based proof-intensive mathematics courses, allows students to easily input mathematical expressions, apply proof strategies, verify logical inference, and apply mathematical rules. Each course has its own language, database of theorems, and mathematical inference rules. The main goal of the project is to create a system that imitates standard mathematical practice in the sense that it allows for natural modes of reasoning to generate proofs that look much like ordinary textbook proofs. Additionally, the system can be applied to an unlimited number of proof exercises.

1 Introduction

The Education Program for Gifted Youth (EPGY), at Stanford University, is an ongoing research project developing computer-based courses and offering them to students via distance learning. We offer courses in mathematics and other subjects and target pre-college students of high ability. The EPGY Theorem-Proving Environment is a tool used in EPGY's proof-intensive mathematics courses. Whereas other computer tools for teaching mathematics (for example, graphing calculators and "dynamic geometry tools") emphasize experimental and inductive approaches, the EPGY theorem-proving environment aims to preserve the traditional emphasis on deductive reasoning in mathematics. In doing so, the system aims to come as close as possible to "standard mathematical practice", both in how the final proofs look and in the kinds of methods used to produce them. In particular, we expect the student to make the kinds of steps normally present in student proofs. The system works to verify the students' logical reasoning and generate and prove "obvious" side conditions that are needed for a correct formal proof but which are routinely omitted in standard practice.

Our use of the Theorem Proving Environment emphasizes teaching mathematics, and we strive to avoid, as much as possible, having to teach a complete logic course or to require lengthy tutorials in how to use a specialized tool. Also, students need to transition in and out of the mainstream curriculum as they move in and out of our program, so we want our courses to look as ordinary

as possible while allowing for the use of the Theorem Proving Environment. Currently the system is being used by students in courses in Euclidean geometry and linear algebra, and it is scheduled for use in courses in calculus and differential equations.

2 Background and Related Work

EPGY is an outgrowth of earlier projects at Stanford in computer-based education dating back to the 1960s. Currently, EPGY offers courses in mathematics from first grade through much of the undergraduate mathematics curriculum. All of these courses are computer-based and self-paced; they consist of multimedia lectures and interactive exercises. These courses are offered to pre-college and college students of high ability as part of a distance-learning program.

Work on interactive theorem proving at EPGY's predecessor, the Institute for Mathematical Studies in the Social Sciences, dates back to the early 1960s with the use of an interactive theorem prover for the teaching of elementary logic to elementary- and middle-school children. With the advent of more powerful automated proof-checking systems, interactive proof systems for university-level logic and set theory were created. These proof systems formed the core component of the Stanford logic course starting in 1972 and the Stanford set theory course starting in 1974 [9].

Work on derivation systems for mathematics courses began in 1985 as part of a project to develop a computer-based course in calculus. This work consisted of three parts. The first part was the formulation of a formal derivation system for differential and integral calculus [2]. The second part was an attempt to integrate the core of the set theory proof system with the symbolic mathematical program REDUCE [4, 10]. The third part focused on interactive derivations of the standard problems in the first year of calculus. Because this system incorporated only a rudimentary knowledge of logic, it was not suitable for proving any fundamental theorems [6].

3 A Description of the Theorem Proving Environment

In the EPGY Theorem Proving Environment, a proof consists of a sequence of proof steps, each of which consists of a mathematical statement and a justification line. Although displayed sequentially, proof steps are represented in tree-form, according to the dependencies of proof steps that are entered as assumptions. Steps that are given initially, entered as assumptions, or taken from the course-specific database of axioms, definitions, and theorems, are immediately recognized as proved. When a newly generated step is shown to follow from already proved statements, it is marked as proved. The initial state of the proof includes a "final goal", and the proof is complete when the final goal is proved. (See the EPGY Theorem Proving Environment User Manual [3].)

The student composes a proof by selectively applying an assortment of tools for creating and justifying proof steps. He or she may apply a logical rule (Section 4.4, below), enter an assumption or apply a proof strategy (Section 3.1), apply a mathematical rule (Section 3.2), or simply type in a new statement and ask the system to verify that it follows from some other step(s) (Section 3.3).

Many of the rules and strategies can be applied in the backward as well as the forward direction. There are typically many rules that can be applied to a particular proof step, allowing for an unlimited variety of paths to a complete proof. Although the system includes a complete set of natural deduction rules, when teaching students to use the system we focus on those that are commonly used in mathematical practice and those that are generally advantageous in our logical framework.

3.1 Proof Strategies

The theorem-proving environment encourages *structured theorem proving*. In particular, students can apply a variety of common proof strategies. These strategies include conditional proof, biconditional proof, proof by contradiction, proof by cases, and proof by induction.

Students can apply these strategies in the forward and backward directions. For example, in order to develop a conditional proof in the forward direction, the student can insert an assumption, proceed to a goal, and then discharge the assumption to obtain a statement of the form “assumption implies goal.” Alternatively, using the conditional proof strategy in the backward direction on a conditional statement generates a conditional proof format where the hypothesis of the conditional appears as an assumption and the conclusion of the conditional appears as the proof goal.

Proof strategies are represented using Fitch-style diagrams, so the student can easily keep track of what assumptions are available at a given location in the proof.

3.2 The EPGY Derivation System

For the application of mathematical rules, the Theorem Proving Environment uses the EPGY Derivation System [6], a tool that has been used in standard algebra and calculus derivations in EPGY courses for over six years. The EPGY Derivation System is an environment in which students can manipulate equations, inequalities, and individual terms by applying built-in mathematical inference rules or term-rewriting rules. The Derivation System is an important tool for generating new proof steps from old ones because it allows the student to flesh-out a claim like “it follows from simple computation that...”

In a typical use of the Derivation System within the Theorem Proving Environment, the student first selects a term or formula he or she would like to manipulate, possibly selects some other proof steps he or she thinks might be

relevant, and finally invokes the Derivation System. Within the Derivation System, the term or formula of interest appears in the “working” area and other relevant facts appear as side-conditions.

Inside the Derivation System, the student essentially manipulates a formula by selecting a subexpression of the “working” expression and invoking one of the rules. (Available rules are determined by the student’s location in the course.) Most rules provide simple algebraic manipulations of existing terms, but many involve side-conditions. For instance, the student may divide both sides of an equation by x , but only if x is a nonzero scalar. As another example, the student may left-multiply both sides of an equation by a matrix A , but only if A has the correct number of columns. Or a student may replace the matrix B by the matrix BAA^{-1} , provided A is an invertible matrix with the correct number of rows. The Derivation System keeps track of side-conditions and does some checking to decide whether a new condition follows from existing ones. Mostly, though, the System merely reports extra side-conditions back to the Theorem Proving Environment, which treats them as new proof obligations. In this regard, the Derivation System is quite different from a normal computer algebra system.

The Derivation System has only the most rudimentary capacity to deal with quantified variables, and its assumptions about terms’ definedness are very restrictive (in contrast with the larger Theorem Proving Environment; see Section 4). Furthermore, a student has very little flexibility to enter brand-new terms. These simplifications are in line with the idea that the Derivation System should carry out the purely “computational” part of the proof. By virtue of these logical simplifications, though, a good student can move rapidly to transform even complicated terms to reach a desired goal.

3.3 Logical Verification

A student may enter a statement as a goal and propose that it follows from some other steps. In listing the justifications of a statement, the student may include other steps from the proof or choose from a database of axioms, definitions, and theorems. As students progress through the course and learn new axioms, definitions, and theorems, they gain the ability to apply them in their proofs.

When the Environment tries to verify the inference, it calls upon the automated reasoning program Otter [5] written by William McCune¹. The Environment passes Otter the justifications as given statements and asks it to prove the proof step using strategies that seem appropriate. If Otter is successful, the proof step is marked as “provable” from the justifications (or “proved”, if the justifications are themselves proved). If it is unsuccessful, the proof step is unchanged. In some cases, when Otter returns unsuccessfully, the student is asked whether he or she would like to “try harder”. This happens when the system has identified that the goal might lend itself to some alternative Otter strategy.

¹ Given that all EPGY students use the Microsoft Windows operating system, the DOS version of Otter is the most suitable automated reasoning system to back our logical-verification tool.

We are constantly identifying and developing alternative strategies for Otter, so this feature changes frequently.

Our use of Otter is intended to mimic an intelligent tutor looking over the student's shoulder and saying "I agree that this follows simply" without the student having to explain all the painful details. The feature allows a student to make logical inferences that would perhaps involve several steps in a detailed, formal proof. To keep the system from accepting large leaps of logic, we limit Otter by allowing it only a few seconds to search for a proof (typically around five seconds). Our experience shows that very often Otter can verify reasonable inferences using our default strategy and time limit. Of course, success depends on many factors, and it is easy to find proof steps and justifications that seem reasonable but are not verified in the allotted time. We continually tinker with Otter's strategies, trying to refine its ability to verify "obvious" inferences but reject more complicated ones; we intend to use student data to help classify "obvious" statements more precisely.

As described below in Section 4, the Theorem Proving Environment presents the student with a multi-sorted logic of partial terms with some variables ranging over functions and other higher-order objects. Otter's logic, on the other hand, is single-sorted, total, and first-order. So we were forced early on to develop a translation from the students' language in the Theorem Proving Environment into statements in standard first-order logic. Some aspects of the translation are mentioned in Section 4.

A drawback to using Otter (and the incumbent translation) for our automatic verification is that the student is not given any information as to why an inference has been rejected. An inference may be rejected because it represents too big a step of logic, because the justifications do not imply the goal, or because our strategies are insufficient for Otter to complete the verification. Merely examining Otter's output, it is difficult to decide which is the reason for a failure. We will examine student data to try to classify common rejections and then program the Environment to give advice. For now, however, the expectation is that the student will study the inference, perhaps break it down into smaller steps, perhaps add some justifications, and perhaps change the statement being verified.

4 Logical Framework

Students using the EPGY Theorem Proving Environment work in a multi-sorted logic of partial terms with function variables, built-in operations on functions, and overloading of function symbols and relation symbols. Each course that uses the theorem proving environment has its own types, function and relation symbols, and conditions for definedness of functions. We have a proof that this logical framework is sound, modulo the soundness of Otter and the correctness of the other external systems that we use for the automatic verification of proof obligations, as described below.

4.1 Definedness and the strong interpretation

For each function symbol included in the language for a course, the system requires an author-coded definedness condition. Relations on terms are interpreted in the strong sense; in particular, the relation

$$R(\tau_1, \dots, \tau_n)$$

as expressed in the “display language” of the students’ world, is understood internally to state

$$R(\tau_1, \dots, \tau_n) \& \tau_1 \downarrow \& \dots \& \tau_n \downarrow$$

where $\tau \downarrow$ is the formula obtained by unraveling the definedness conditions of the functions in τ .

4.2 Quantifiers and types

Since the Theorem Proving Environment uses a logic of partial terms, quantified variables range only over defined terms. Furthermore, it uses a multi-sorted language to ensure that quantified variables range only over terms of a particular type. For instance, in a linear algebra course, where “ A ” is a variable whose sort is “matrix”, a theorem about matrices might be stated “ $\forall A \phi(A)$ ”. To instantiate such a theorem, a student would need to provide a term denoting a matrix: a term such as “ $1 + 2$ ” would not be allowed at all because $1 + 2$ is not a matrix.

Sorts are not the only tool for describing types; there are explicit relations, too. For each type T , the Theorem Proving Environment includes a relation “is a T ”. The type-relations sometimes appear explicitly as proof obligations. Most type conditions are purely syntactic and are automatically either accepted or rejected by the system. Some are harder to decide and are left to the student as proof obligations (Section 4.3). An example of a “harder” type condition is deciding whether or not $(a + bi)(c + di)$ is a real number, where $a, b, c,$ and d are reals. To justify the proof obligation “ $(a + bi)(c + di)$ is a Real”, the student might invoke a theorem that this is true when $ad + bc = 0$.

There is one other important use of our type-relations. When theorems are passed to Otter as justifications or goals, they must be translated from the multi-sorted language of the Theorem Proving Environment into something Otter can understand. The usual translation of the theorem “ $\forall A \phi(A)$ ” is “ $\forall A (A \text{ is a Matrix} \rightarrow \phi'(A))$ ” (where ϕ' is a translation of the formula ϕ).

4.3 Proof obligations

In enforcing the strong interpretation of relations, the Theorem Proving Environment generates many extra formulas, or “proof obligations”, which the student must justify in order to complete his or her inferences. Many such formulas are simply added to the proof as new proof lines. For instance, if a student wants to instantiate “ $\forall A \phi(A)$ ” with the matrix B^{-1} , he or she will need to prove that

B is an invertible matrix. This is simply added as a new proof goal; it is the student’s responsibility to justify the statement.

Whenever a new proof obligation is generated, the Environment does some simple checking to see whether it is “obvious” before adding a new proof step. It may call Otter or do some other internal processing. The purpose of this checking is to keep the proof from becoming cluttered with simple, obvious facts. We think of such obvious facts as being the type of “hidden assumption” that is often wrapped up in a statement in an ordinary mathematics textbook.

4.4 Proof rules

The Theorem Proving Environment provides several rules that create and justify proof steps. The student can instantiate a quantified statement, use generalization in the forward or backward direction, and substitute for equals or expand definitions in the forward or backward direction. In this section, we describe some of these rules, focusing on how they relate to our multi-sorted logic of partial terms.

Universal instantiation is a fairly simple rule. Using a statement $\forall x \phi(x)$, the student supplies a new term τ to conclude $\phi(\tau)$. As part of this action, the Theorem Proving Environment may require extra proof obligations about the definedness or type of τ .

The student may use universal generalization in either the forward or the backward direction. From a statement

$$\phi(x), \tag{1}$$

where x is a free variable, the student may conclude

$$\forall x \phi(x). \tag{2}$$

The student is working “backwards” if he or she starts with statement (2) and creates statement (1); in this case, step (2) is marked “Provable assuming (1)” and will become “Proved” when (1) is justified. The student is working “forwards” if he or she starts with statement (1) and generates statement (2) as a conclusion.

Existential generalization and existential instantiation are dual operations to universal instantiation and universal generalization, respectively. They may generate appropriate proof obligations, and existential generalization can be used in either the forward or the backward direction.

Students have two other tools for creating new proof lines from old ones. Using the “substitute” rule and a statement of the form $\gamma \rightarrow \tau = \sigma$ (or a similar statement with a quantifier), the student can create a new proof line $\phi(\sigma)$ from the line $\phi(\tau)$. The Environment will require γ as a proof obligation. The “expand definition” rule works similarly, except that it starts with a statement of the form $\forall \vec{x} (R(x_1, \dots, x_n) \leftrightarrow \phi)$.

When using substitution or definition expansion from a quantified statement, we must be careful not to apply the rule to an undefined term. So, for instance,

if the student tried to use the substitution rule $\forall x (x \neq 0 \rightarrow \frac{x}{x} = 1)$ to make a substitution from $\frac{\tau}{\tau}$, the Environment would generate proof obligations from both $\tau \neq 0$ (the hypothesis of the formula) and $\tau \downarrow$ (to account for the instantiation).

4.5 Higher Types

As mentioned earlier (Section 3.3), the Theorem Proving Environment uses Otter to check some verifications, and Otter uses a first-order language. The language of the Theorem Proving Environment, however, has variables ranging over functions and other higher types. Some aspects of this higher-order logic are easy to translate for Otter, but others are more difficult.

Simple function-valued variables are handled relatively easily. We simply add an “apply” function² to the language of the Theorem Proving Environment. The definedness condition of a term “apply(f, x)” is essentially “ x is in the domain of f ”, so the Theorem Proving Environment has such a binary relation³. Given these operations, functions become first-class objects, so it is easy to translate them for Otter. Since we control the vocabulary of the Theorem Proving Environment, we can handle explicit higher-order functionals similarly.

We also handle the more complicated case where an operation on functions is applied to an open term that implicitly defines a function. For example, the simple term “ $\sum_{k=1}^n A_{k,k}$ ” (defining the trace of the $n \times n$ -matrix A) is interpreted as “sum($1, n, f$)”, where f is the function defined by $f(k) = A_{k,k}$. Other expressions, like “the $m \times n$ -matrix whose (i, j) -th entry is $\frac{1}{i+j}$ ” and “ $\frac{d}{dx} \sin(x)$ ”, require functions implicitly defined by terms. As part of its translation for Otter, our system needs to extract these functions and give them explicit descriptions. As an illustration, we give the details of how summation is translated.

Example: Summation. In translating the formula $\phi(\sum_{k=m}^n \tau(k))$, we assume that the term τ does not contain any summations. We will replace the summation-term with an equivalent one without changing the rest of the formula.

We create a new function-variable f and add the axioms

$$\forall k (\tau(k) \downarrow \rightarrow f(k) = \tau(k)),$$

$$\forall k (\tau(k) \downarrow \rightarrow k \text{ is in the domain of } f)$$

(where “ $\tau(k) \downarrow$ ” represents the formula obtained by unraveling all the definedness conditions for functions appearing in $\tau(k)$). Then the original formula can be replaced by $\phi(\text{sum}(m, n, f))$. The definedness condition for the term $\text{sum}(m, n, f)$ function is

$$\forall k (m \leq k \leq n \rightarrow k \text{ is in the domain of } f).$$

² Technically, we need a separate apply function for unary functions, binary functions, etc., because Otter does not support functions with variable numbers of arguments.

³ Technically, binary, ternary, etc. relations.

In fact, this is an unwieldy translation, especially if the formula contains many nested summations (we work inside-out in that case). Otter can directly verify only the simplest statements about summation, so the student typically needs the Environment's other tools to write proofs about summations.

5 Student Use

Presently, the Theorem Proving Environment is being used in EPGY's geometry and linear algebra courses, and it will be incorporated into the EPGY multi-variable calculus and differential equations courses later this year. The proof environment has been used by approximately 60 students in the geometry course.

In our geometry course students are able to prove theorems about incidence, parallel, betweenness and congruence (including some of the basic triangle congruence theorems) in a version of Hilbert's axioms of geometry. In the linear algebra course the system is capable of theorems in matrix algebra and theorems about eigenvalues and eigenvectors. In calculus, the system can be used for basic continuity theorems; for example, that the sum of two continuous functions is continuous.

The history of the students' actions are recorded so that we can "play back" a student's proof from start to finish, displaying all steps including those that were deleted by the student. Using this feature, we have had the opportunity to examine many students' proofs. The analysis of these proofs has influenced both the development of the system and our presentation of the material in our courses.

6 Future Directions

The main goal of the Theorem Proving Environment project is to develop a system that imitates, both in construction and final form, the proofs of "standard mathematical practice". For this reason, many of our directions of future study will focus on how to make the Theorem Proving Environment a more natural tool for students.

One clear area to target for improvement is the system's ability to automatically discharge "obvious" proof obligations. Because of the way proof obligations are generated, it is often the case that another statement within the proof would be helpful as a justification. Classifying exactly when a statement is "relevant" will be a very difficult task, but some initial experiments have shown that we can improve Otter's performance by first scanning through the proof with some very basic algorithms.

As a further aid in discharging proof obligations, we are studying more extensive use of computer algebra systems. We currently use Maple V, Release 5.1, to automatically check some obligations that appear to involve only algebraic computations. This has been a successful strategy so far, but one drawback is that most computer algebra systems (Maple included) were designed for efficient computation and not necessarily for logical soundness so we need to be

very careful when we use them. Nevertheless, when used appropriately, Maple in particular can quickly decide many questions which give Otter great difficulty.

Given that proof obligations are often generated in a restricted, decidable fragment of a mathematical theory, we intend to investigate the use of decision procedures for these cases. For example, in linear algebra and analysis, it is common to generate proof obligations that are expressed as order relations on linear terms with integer coefficients. Additionally, we may incorporate such decision procedures into the verification process directly, in cases where we can automatically determine that they apply.

In a further effort to make our students' proofs more natural, we would also like to improve our treatment of higher types. At the moment, our built-in tools deal fairly well with functions and operations on functions. When we need to translate higher-order objects for Otter, however, the statements quickly become very complicated, so some seemingly simple statements are hard to verify. This will be an increasing problem as we develop our differential calculus course. We plan to continually improve and simplify our translation, but we will also consider using provers based on higher-order logic. In principle, our system can use any number of automated reasoning systems for logical verification; however, given the limitation resulting from our use of Microsoft Windows, most higher-order provers (e.g., PVS, HOL, ACL2, etc.) are not available for our immediate use.

References

1. Barwise, B. & Etchemendy, J. (1999). *Language, Proof and Logic*. Seven Bridges Press. New York.
2. Chuaqui, R. & Suppes, P., (1990). An equational deductive system for the differential and integral calculus. In P. Martin-Lof & G. Mints, (Eds.), *Lecture Notes in Computer Science, Proceedings of COLOG-88 International Conference on Computer Logic (Tallin, USSR)*. Berlin and Heidelberg: Springer Verlag, pp. 25-49.
3. Education Program for Gifted Youth (EPGY). Theorem Proving Environment Overview. <http://epgy.stanford.edu/TPE>.
4. Hearn, A. (1987). Reduce user's manual, Version 3.3. (Report CP 78). The RAND Corporation. Santa Monica CA.
5. McCune, William. OTTER 3.0 reference manual and guide. Technical Report ANL-94/6. Argonne National Laboratory. January 1994.
6. Ravaglia, R. (1990). *User's Guide for the Equational Derivation System*. Education Program for Gifted Youth, Palo Alto.
7. Ravaglia R., Alper, T. M., Rozenfeld, M., & Suppes, P. (1998). Successful Applications of Symbolic Computation. In *Human Interaction with Symbolic Computation*, ed. N. Kajler. Springer-Verlag, New York, pp. 61-87.
8. Sieg, W., & Byrnes, J. (1996). Normal Natural Deduction Proofs (in classical logic). Tech-report CMU-PHIL-74. Department of Philosophy, Carnegie Mellon Univ., Pittsburgh, PA 15213.
9. Suppes, P. (Ed.). (1981). *University-level computer-assisted instruction at Stanford: 1968-1980*. Stanford, CA: Institute for Mathematical Studies of the Social Sciences, Stanford University.
10. Suppes, P. & Takahashi, S. (1989). An interactive calculus theorem-prover for continuity properties. *Journal of Symbolic Computation*, Volume 7, pp. 573-590.