Using Web Access to Formal Mathematics to Support Instruction in Computational Discrete Mathematics

Robert L. Constable

March 7, 2002

Abstract

We have begun a project to produce *interactive formally-grounded courseware* for teaching mathematics and computing.[1] The courseware is created by a modern proof development system, Nuprl, based on its growing em reference library of formal computational mathematics. The project is supported by NSF and some results of the past eighteen months of work are available on the World Wide Web.[2] This proposal requests an increment of funding to supplement the continuing investment of Cornell resources.

We are asking for funds to improve the educational value of the resources we have created. First, we want to add more *targeted lessons* as entry points to the large corpus of formal material. Second, we want to gather feedback on the existing lessons from a wider range of students and instructors. Third, we want to prepare for using the full Nuprl interactive capability when it becomes available on the Web in 1998 and then deploy it in 1999.

This proposal reviews the technical and pedagogical case for the project, reports on current progress and future plans and explains our ideas for improving the educational value of the material already created.

[1]NSF grant DUE 9555162 Creating and Evaluating Interactive Formal Courseware for Mathematics and Computing. The project is called Interactive Formal Courseware

[2]The url is www.cornell.edu/Info/Projects/NuPrl/html/Interactive_Formal_Courseware.html.

Contents

1 Introduction **3**

1.1 Brief Overview 3

1.2 Previous Work 7

1.3 Proposed Work 8

2 Sample of Formal Reference Mathematics and Lesson **9**

2.1 Research Technology in Education 9

2.2 Example of Reference Material 10

2.3 Hypertext, Terms, and Proofs 11

2.4 Sample Lesson 13

3 The Nuprl Computational Mathematics Medium **20**

3.1 The Formal Language 20

3.2 Tactic-style Theorem Provers 21

3.3 Built-in Functional Programming Language 22

3.4 Library Structure and Use 23

4 Educational Aspects **24**

4.1 Problems and Opportunities 24

4.2 Experience with Students Learning Discrete Mathematics and Logic 25

4.3 Evaluation and Monitoring 26

4.4 Exploration and Collaboration 26

4.5 Criticisms 27

5 Work Plan **28**

5.1 Background 28

5.2 Schedule and Milestones 28

A An Exercise in Induction **40**

B Summary and Impact **45**

Results From Prior NSF Support

The project started with NSF grant DUE-955162, duration January 1996-December 1997. Narrative about previous results appears in section 1.2 Previous Work of the proposal. Basically, we have created a large body of formal courseware and some targeted lessons built on it. We designed and implemented software to present this material on the Web to be viewed with Netscape and Explorer. The material was used in Cornell courses and is now available to be used elsewhere. There were two publications [21, 24] in the eighteen months since we have been funded plus three "Web publications."

The course material can be accessed at the following url:
www.cornell.edu/Info/Projects/NuPrl/html/Interactive_Formal_Courseware.html.

1 Introduction

1.1 Brief Overview

goals and background The heart of the Interactive Formal Courseware Project is a plan to explore the value of *formally-grounded explanation* in improving students' understanding of mathematics, especially of computational discrete mathematics. By a formally-grounded explanation we mean one that ultimately can be reduced to a readable machine checked proof in a formal logic. The entire project rests on the results of over twenty-five years of research that produced a new generation of reasoning tools such as Nuprl (and other similar systems under construction worldwide [60, 25, 43, 53, 61]). Later (in section 3) we will briefly describe this research, but for the moment let us accept that these tools enable us to produce formal definitions, theorems and proofs covering most of the subjects taught in college level discrete mathematics. Let us also assume that this formal material can be read and understood by mathematically educated first and second-year college students in a large number of the nation's 2,000 four-year colleges and universities. The reader will be able to judge these matters by reading the proposal.

pedagogical issues What is the educational value of formally-grounded explanation, and what known pedagogical problems does it solve? The brief answer to the first question is that this novel courseware opens significant new opportunities for using computers in education, especially for bringing mature research software into learning via the Web. The brief answer to the second question is that we already have shown from experience with the first versions of this material that it solves known problems with traditional mathematical texts, that it helps overcome documented difficulties teaching mathematical problem solving, and that the specific courseware we are creating reinforces ties between mathematics and computing that are known to help teach the basic concepts of function, induction and proof that are fundamental and yet problematic to teach. The proposal takes up these answers in detail in section 4, here I elaborate on them briefly.

First let us consider the new opportunities. Notice that because explanation is so fundamental to education, every potential advantage in teaching it is heavily weighted. Whether we are talking about an English essay, a chemistry experiment, a legal argument or a mathematical demonstration, college students are taught to answer the question "How do you know?" They are taught to give evidence and to say what statements follow from others. This ability to provide evidence and evaluate arguments is critical to a liberal arts education or an engineering one.

The added value of formal explanation is that it can be manipulated by machines as well as people, it can be automatically linked to semantically related material, and it can be related to a foundational account of mathematics and computing. These are enormous advantages as will be clearer when we illustrate them in section 2. Computers help create these explanations and check them for errors and omissions, but they can perform numerous other functions for us—showing exactly what other concepts a given concept depends on, "what depends on what", and finding semantically related concepts.

The underlying foundation is also very significant [19, 35, 74], especially for connecting computing and mathematics. In programming, students understand that meaning is ultimately given in terms of simple computational procedures that people and machines follow in essentially the same way. This "semantic agreement" is the basis of all mathematical properties of programs from speed to correctness. This agreement then is the foundation of knowledge about programs, and it can be used to help teach about functions in mathematics.

The semantic paradigm for programming extends in principle to computational mathematics as well. For example, we can explain that numerical expressions, exp_1 and exp_2 are equal, $exp_1 = exp_2$, when both compute to the same number.

The second question of this section is what pedagogical problems the formally-grounded courseware solves. The answer will be clearer after we have examined the formal reference material more closely, but consider first the issue of standard mathematics textbooks. Most of them do not bear careful logical scrutiny, at best they contain small but annoying errors, at worst they contain major conceptual errors. Even the best of them suffer from errors and omissions of detail that students waste countless hours puzzling over. Reference books and textbooks alike exhibit the problem of locating key information, such as definitions, notations or theorems. Good texts have large indexes, but even they are tedious to use and often fail. The problem of missing motivation to proof steps leaves readers to wonder why a simpler justification they have in mind is inadequate. The courseware we produce alleviates these problems.

Studies [1] have shown that students have difficulty understanding the goal and subgoal structure in solutions to problems, especially those solutions given by induction. This is also clear from books dvoted entirely to proof [26, 77, 69] and studies in logic [65, 51, 39]. The proof structure we adopt is excellent for alleviating this difficulty as we show. Finally, we have already noted that relating functional programs to mathematical functions is known to help with the problem of teaching functions [42, 1]. Our reference material is especially suited to relating computational and mathematical concepts, and we exploit this capability in our lessons.

technical experience Since 1975 we have been building *proof development systems* — computer systems that help people create formal proofs. The latest version is Nuprl 4.[3] These have been built largely with NSF funding, and they are available without charge. Nuprl is essentially the first major *theorem prover* to treat proofs as mathematical objects; proofs are central to the system architecture, to the underlying logic, and to the proposal.

Nuprl is a successful research tool which has been used to generate libraries of highly readable formal mathematics in a variety of subjects [63, 33]. Our research group at Cornell has extensive experience with this medium for communicating technical ideas. Nuprl has been used for hardware verification [48], for programming language semantics and as a programming logic [22, 45, 21], for supporting symbolic algebra [46], for building components of theorem provers [23], and for manipulating communication protocols.

We taught formal mathematics using Nuprl in three advanced courses. The idea was that the Nuprl libraries were the underlying reference material for the course. Out of this experience came the idea for formally-grounded courseware at the heart of this project.

obstacles to formalization overcome As a consequence of our technical accomplishments, we have overcome a number of commonly recognized obstacles to the use of formalisms to naturally express and teach mathematics.

- The use of normal-looking, flexible notations supports our habits of thought, which

[3]The systems have been λ PRL, (lambda pearl), Nuprl ("new pearl"), Nuprl 3 and now Nuprl 4.

could be rendered ineffective by the need to mentally translate back from a clumsy formal notation (see section 2.2).

- The capacity to suppress detail in expressions is key to simplifying our reading of them. This is really a corollary to the previous point, since examination of the informal notations we use reveals a lot of elision.
- Formalization need not be complete to be useful, and indeed, formal and informal expressions can be mixed.

However important these may be to experts, they are crucial to students. The benefits of formalization for beginners could have been ruined by such obstacles.

educational opportunities The interactive formal hypertext data base is useful in teaching *facts, techniques* and certain modes of *understanding.* Here is how.

1. The level of detail at which students interact with the material is *flexible*, from high level summaries and guided readings down to an examination of *every detail* and every lemma required for a proof. We can collect information about how students use the system and gain quantifiable insights into learning behavior.

   Often students are frustrated because a step in an argument that was trivial to the author is not clear to them or because some critical fact was left implicit. Since these proofs are complete, all the details are present, but on demand.

2. Providing multiple *readings* of a proof or a library section allows the instructional staff to offer many different approaches to the same idea. So there are *diverse entry points* to match a diverse student body. Using CoNotes, students will be able to contribute their own commentary directly in the library or can annotate those we provide. This personalizes the course material and encourages *collaborative learning.*

3. The mechanisms of a tactic-style prover allow us to teach *problem solving technique.* The vocabulary of *goals, subgoals, rules, tactics,* and *lemmas* enables us to discretize and quantify some of the learning [66]. This discovery of tactics [34] and their refinement to *tactic-tree proofs* is not only a major advance in formal methods, it may also be a major contribution to pedagogy, as we intend to examine further.

4. Since Nuprl's formal language can express virtually any mathematical concept, we can begin to *teach understanding* as a process of making connections between topics. As

we make accessible our research libraries through lessons, we will be able to relate such ideas as induction and recursion, algebraic structures and program modules, graphs and relations, etc. In addition to the connections mentioned with *existing libraries*, we could relate Boolean rings, the propositional calculus, digital circuits, monoids and polynomial rings.

impact and significance We know from direct experience and from the literature [73, 39, 40] that some of the fundamental concepts in mathematics, such as *function, induction* and *proof* are difficult to teach. These concepts are central in both continuous and discrete mathematics, and the function concept seems to be critical in understanding *abstraction*. These concepts are basic to the language of modern science and engineering. They are as important in computer science as they are in physics. Improvements in education of the kind we suggest will help the U.S. retain its technical leadership and may help open the sciences and engineering to a wider segment of the population.

The computer-aided instruction (CAI) tools now available are mainly limited to first order logic, algebra, and geometry *in isolation* and deal principally with techniques [11, 3, 5]. We propose a way to enhance deep understanding as well as technique and factual knowledge.

1.2 Previous Work

The initial proposal submitted in 1995 requested three years of funding to create a repository of material for mathematics and computer science education as well as funding to disseminate and evaluate the materials. We were funded for two years with the goal of creating course material. Now, eighteen months into the project, we have produced the first version of most of this body of material.

The courseware project began to unfold just as the World Wide Web exploded on the national scene. The emergence of the Web substantially altered our plans. We recognized the imperative that our material be made available on the Web. We designed and built software which produced html code for any Nuprl library and put these libraries on our Web site. We created an example of integrated formal and informal material by formalizing sections of automata theory from Hopcroft and Ullman's classic text book *Formal Languages and Their Relation to Automata*. We also made available material on logic based on Smullyan's approach in his superb text book *First-Order Logic*.[4] The Web-based material includes these new documents and libraries

[4]We also persuaded Dover to publish this classic work which had gone out of print at Springer.

1. *Formalizing Automata Theory I: Finite Automata*

2. *Extracting Propositional Decidability: A proof of propositional decidability in constructive type theory and its extracted program.*

3. *Simple Imperative Programming Languages*

The results from 1 are being published [24], and the overall project was the subject of the publication [21] from the NSF Showcase Project. The PI spoke on this subject at the 1996 Frontiers in Education conference as an NSF Showcase Project.

1.3 Proposed Work

overview We propose to improve the educational value of the resources we have created in three ways: implement more formally-grounded lessons and improve the existing ones in response to feedback, link Nuprl Web material to the Cornell/Xerox CoNote system to enable students to comment directly on the lessons, and build interactive lessons in Nuprl that can be released on the Web as the Nuprl research group moves more functionality onto the Web. We elaborate these points.

lessons In section 2 we present part of a lesson built on top of a library called `functions_1`. The appendix shows a lesson on induction grounded in the library `number_theory_1`. There are other lessons at the Web site, but we need to create several more to take advantage of the existing libraries of discrete mathematics and to provide instructors of various kinds of courses with material. In particular, we intend to produce lessons for discrete mathematics (CS, Math, EE), for logic (CS, Math, Philosophy), automata and formal languages (CS, Linguistics). We aim to create two special lessons, one on Gödel's theorem which can be so nicely presented in the Nuprl logic, and one on Automata theory, specifically state minimization.

student feedback The CoNote system is software developed by Cornell and the Xerox Design Research Institute at Cornell to enable a group of users of html documents to share annotations of the document. By adding CoNote anchor points to our lessons and providing a CoNote server with the Web material, readers of the courseware will be able to add comments and questions directly to the lessons. We can use this capability to enhance the value of the lessons by leaving the good annotations in place (like a frequently asked questions, FAQ, capability), and we can use it to collect feedback for improving the lessons themselves. We could even assign students and staff to monitor lessons on a weekly basis and answer the frequently asked questions. A small amount of funding would enable this (one hour a week for five students for an academic year—see FAQ response in the budget).

interactivity Students using the Nuprl system itself would have access to a wide range of interactive capabilities. They can create proofs or modify existing ones. They can be led through new proofs. They can solve exercises. They can step through rewrites and other symbolic computations step by step as well as stepping through ordinary compuations.

Throughout 1998 the Nuprl research group will provide more of the full interactive capability directly on the Web. This will allow us to put the highly interactive lessons in the Courseware Library. (For the sake of this proposal we could call this NuprlW.)

2 Sample of Formal Reference Mathematics and Lesson

2.1 Research Technology in Education

tools for the classroom We know that it is possible to use the latest technology in teaching, for example, Macsyma was available to undergraduates at MIT on Multics already in the 70's, and *Mathematica* [81] and Matlab are now widely used. CAD tools are used in hardware courses and so on. In modern computer science we see comprehensive programming environments used in the classroom; it would even be possible to use program verification tools.

Essentially we are proposing to gradually move a high-powered research tool, Nuprl, into the classroom. The first steps are possible because of the Web and because PC's under Linux and Sun's under Solaris can run Nuprl for people willing to install it. Also, more functionality of Nuprl will be made available via the Web over the next eighteen months. Underlying the use is a close alignment between the aims of the research and the needs in the classroom. The devices we have built for our Formal Methods research are precisely what is needed to teach mathematics and computing.

emergence of formal mathematics Another factor influencing this proposal is that there has been a dramatic rise in the amount of formal mathematical material over the past decade.[5] In the 1970's it took several years to formalize Landau's very small book *Grundlagen der Analysis* [49] in Automath. Now we could generate that material in a matter of weeks. The Mizar project has published twelve volumes of computer-checked mathematics

[5]This is not all "text" since some is stored only electronically.

in its *Journal of Formal Mathematics* covering over 300 articles authored by over 66 mathematicians. Nuprl alone is responsible for over 40 article-size pieces. We are proposing to organize some of this into course material which can be done in an especially instructive way since Nuprl generates readable proofs — whereas most other systems accept or reject conjectures without readable justification (people call these "write only proofs").

2.2 Example of Reference Material

functions We have chosen to illustrate the underlying body of formal mathematics on a fragment of the basic theory of functions. The fragment commonly appears in textbooks on discrete mathematics CHECK:[36, 37, 2]. This is a topic for which very *good formalizations* exist in the sense that informal methods can be faithfully formalized without prematurely introducing distracting details of logic. COMPARE WITH OLD

Nuprl library example A Nuprl *library* is a sequence of various kinds of objects, including definitions, theorems and comments. Libraries are divided into sections called *theories.* Here we give a partial listing of a Nuprl theory covering covering basic facts about functions, showing the definitions and theorems. Below we shall show how a student would access this material via explanatory material; here we just show the kind of formal data in which such material is grounded.

We begin with a few fundamentals before introducing new concepts. In the lines below, the `T` at the line start indicates that this object is a (potential) theorem. The `*` before the `T` shows that the theorem has been completely proven. (If the proof is not complete, the `*` is replaced by a `#` character).

```
*T fun_extensional       ∀ f,g:A→B. f = g⇐⇒(∀ x:A. f(x) = g(x))
*T ax_choice             (∀ x:A. ∃ y:B. Q(x;y)) ⇒ ∃ f:A→B. ∀ x:A. Q(x;f(x))
*T fun_description       (∀ x:A. ∃! y:B. P(x;y)) ⇒ ∃ f:A→B. ∀ x:A. P(x;f(x))
```

The first of these expresses the fact that identity on functions depends only on input/output, the second is the axiom of choice, and the third is the weaker claim that a function is determined by describing how a unique output is related to each input. The definition of two functions being 'inverses' is:

```
*A inv_funs              InvFuns(A;B;f;g) == (∀ x:A. g(f(x)) = x) & ∀ y:B. f(g(y)) = y
```

Here, the `*` shows that the object is complete and the `A` shows that this is an *abstraction*, Nuprl's name for a definition. Next are a couple of basic theorems about inverse functions, namely that an inverse of a function has that function as its inverse, and that a function can have at most one inverse.

```
*T inv_fun_sym           InvFuns(A;B;f;g) ⇒ InvFuns(B;A;g;f)
*T inv_fun_unique        InvFuns(A;B;f;g) ⇒ InvFuns(A;B;f;h) ⇒ g = h
```

Here are the definitions of 'injection', 'surjection', and 'bijection', and some standard theorems:

```
*A surjection            Surj(A;B;f) == ∀ b:B. ∃ a:A. f(a) = b
*A injection             Inj(A;B;f) == ∀ a1,a2:A. f(a1) = f(a2) ⇒ a1 = a2
*A bijection             Bij(A;B;f) == Inj(A;B;f) & Surj(A;B;f)

*T linv_of_surj_is_rinv  Surj(A;B;f) ⇒ (∀ x:A. g(f(x)) = x) ⇒ ∀ y:B. f(g(y)) = y
*T fun_with_inv_is_bij   InvFuns(A;B;f;g) ⇒ Bij(A;B;f)
*T bij_imp_exists_inv    Bij(A;B;f) ⇒ ∃ g:B→A. InvFuns(A;B;f;g)
```

The concept of one-to-one correspondence is defined in terms of inverse, and is shown equivalent to bijection.

```
*A is_one_one_corr       f is 1-1 corr == ∃ g:B→A. InvFuns(A;B;f;g)

*T bij_iff_is_1_1_corr   Bij(A;B;f)⟺f is 1-1 corr
```

A lesson about one-to-one correspondence and these definitions may be found below. We'll end this sampling of the library with the concept of function composition. These definitions define a function symbol by showing how to rewrite the application of the function symbol to a generic argument.

```
*A compose               (f o g)(x) == f(g(x))
*A identity              Id(x) == x

*T comp_id_r             ∀ f:A→B. f o Id = f
*T comp_id_l             ∀ f:A→B. Id o f = f
*T comp_assoc            ∀ f:A→B, g:B→C, h:C→D. h o (g o f) = (h o g) o f
*T comp_preserves_surj   Surj(A;B;g) ⇒ Surj(B;C;f) ⇒ Surj(A;C;f o g)
*T comp_preserves_inj    Inj(A;B;g) ⇒ Inj(B;C;f) ⇒ Inj(A;C;f o g)
*T comp_preserves_bij    Bij(A;B;g) ⇒ Bij(B;C;f) ⇒ Bij(A;C;f o g)
```

2.3 Hypertext, Terms, and Proofs

Typical lesson material consists of documents blending formal with informal text which are intended to be read with an interactive browser. One particularly useful hypertext link is one that points to a proof of a theorem. One may, of course, create an ordinary hypertext link to a proof, but often it is more convenient to show the content of the theorem at its point of use, and so Nuprl includes links that display content. For example:

```
THM* InvFuns(A;B;f;g) ⇒ Bij(A;B;f)
```

When the user clicks on this, the formal proof, or a document informally glossing it, is opened for viewing. Such a link to a theorem may be embedded in informal hypertext, and may also be used within proofs of other theorems that use it as a lemma. How formal proofs are read will be discussed in the next section. Another example of these links that display content are the references to definitions such as:

```
DEF Surj(A;B;f) == ∀ b:B. ∃ a:A. f(a) = b
```

Not only do links of this kind enable the user to track down references, but they also prevent documentation from getting out of "sync" should the objects or notations get modified.

definitions and blending materials The ability to find the definition of an operator from any use of it throughout the system is a key feature of the Nuprl editor. For example, if the user wanted to see the definition of "divides," he or she could click on any occurrence of it, such as "`n·a | n·b`", and the defi nition would be viewed, which looks like this:

```
b | a == ∃ c:ℤ. a = b·c
```

It is also possible to associate other documentation with the definition which will be presented when the definition is viewed, such as informal descriptions and links to related lessons. Such occurrences of the "divides" operator are formal no matter where they occur, either in proofs or in informal documents. Besides the benefit of being able to find the definition, if the user wants to change the way an operator is displayed, say to "`x divides y`" instead of "`x | y`", this change will be automatically adopted throughout the system since it is the abstract expression rather than the text "`x | y`" that is used.

Formal and informal material can be freely mixed. If this proposal had been presented on the Nuprl system, for example, you could actually have clicked on the theorem references and operator occurrences occurring above with the described effects. So one can write informally about a subject but use the formal notations whenever desired, thus allowing the user full access to relevant formal materials such as definitions and proofs, and any other documentation that has been attached to the formal materials. Conversely, one can insert commentary inside the formal objects of the system.

structure editing Formal mathematical notations are edited with a visually oriented structure editor [63]. It is a structure editor rather than a text editor because we edit the abstractly represented term directly rather than a string of characters that gets parsed into a term. The editor is visually oriented in that the interpretation of most edit commands depends not only upon the abstract term structure, but also, of course, on the display forms in use. Most of this editing is performed by means of cut-and-paste operations and various menus, so it is not necessary for the students to master a large "name space" for commands, operators, and library objects.

some conventional notational devices supported by Nuprl It often happens that when you develop a body of mathematics, you find you have implicitly parameterized it by one or more variables held fixed throughout large expressions. In Nuprl, these implicit parameters would fill actual argument places in the underlying term, which would be elided during display; but if filled by some other expressions, these places would be shown.

Another conventional notational device is to abbreviate the display of certain iterated operators.

- `<a,<b,<c,d>>>` is displayed as `<a,b,c,d>`
- `∀ x:ℕ.  ∀ y:ℕ.  ∀ z:ℝ.  P(x;y;z)` is displayed as `∀ x,y:ℕ, z:ℝ.  P(x;y;z)`

These and other standard notational abbreviations are supported by the Nuprl editor through a fairly general scheme.

The larger point is that *we respect conventional notations*, which we must suppose have often developed for good reason. When we can, and can afford to, we try to adopt conventional devices into our system. The obstacles in any case are figuring out how the notation works, and determining whether it is compatible with prior systematic constraints on formal expression we have established for Nuprl.

2.4 Sample Lesson

What follows is an excerpt of a sample lesson on one-to-one correspondence. The full lesson may be found in the appendix. Because of space limitations here, we must elide some of the material, and show only a few pages of it. The starting point for this lesson is the following documentation object, which is mostly informal text with formal mathematical notation embedded.

```
One-to-One Correspondence.

A one-to-one correspondence between two classes is a way of matching them
member-for-member. Examples:

  At an event with assigned seating one would expect a one-to-one correspondence
  between the seats and the tickets, the correspondence being between each seat
  and the ticket printed with its location code.
  ...

  A major use for finding one-to-one correspondences is to establish, without
  actually counting, that some class has the same size as another class of known
  size.

  ...

Notice that a one-to-one correspondence between two classes will not be the only
one if the classes have more than one member each.

As usual, this concept of a method for matching is not built-in to our growing
library of mathematics, and may be introduced by definition in a variety of
ways.

We shall use functions for these methods of matching: a function will be used to
match each input with its output. For example, the function
f ∈ℕ→ℕ such that f(n) = 2·n, matches each natural number with its double, and
is a one-to-one correspondence between the natural numbers ℕ, and the even
natural numbers, {i:ℕ| i Even}.

...
```

From here, the reader continues to a discussion of inverse functions:

```
One-to-One Correspondence: Inverse Functions

As was discussed ELSEWHERE, we are trying to characterize the functions which
determine one-to-one correspondences. We shall give two such "definitions".

One observation is that when a function f ∈A→B matches A and B
member-for-member, then one can reverse this function to get some g ∈B→A which
matches these classes in the opposite direction. When functions f and g have
this inverse relation between them, we say they are "inverses" and write
InvFuns(A;B;f;g). We define this relation between functions as

  DEF InvFuns(A;B;f;g) == (∀ x:A. g(f(x)) = x) & ∀ y:B. f(g(y)) = y

  Notice that this inverse relation between functions is symmetric, i.e.,

  THM* InvFuns(A;B;f;g) ⇒ InvFuns(B;A;g;f)
  (just swap the conjuncts in the def)

Cancellation:
  When in the course of reasoning one uses the fact that (g(f(x)) = x) to
  rewrite g(f(x)) to x, this is sometimes called "cancellation" of f by g.
  Inverse functions cancel each other. For example, subtracting an integer (from
  something) and adding that same integer are inverse functions, so you may use
  one to cancel the other, by rewriting x-a+a or x+a-a to x.

Uniqueness:
  As you can imagine, if a function has an inverse, that inverse is unique.
  Here is a theorem to that effect:

  THM* InvFuns(A;B;f;g) ⇒ InvFuns(A;B;f;h) ⇒ g = h

  As usual, some persons who have doubts about our choice of definition for
  InvFuns(A;B;f;g) might use this fact as further justification of the
  definition; whereas those who already find the definition adequate might use
  the proof either to complement the imagination, or resolve doubts about
  uniqueness.

We adopt the following definition of one-to-one correspondence:

  DEF f is 1-1 corr == ∃ g:B⟶A. InvFuns(A;B;f;g)

There is another characterization of one-to-one correspondence, involving
"bijection", which may well seem more obviously right. To follow this
development, click HERE.
```

The various references to definitions and theorems are special links that show the actual content of the proofs and definition objects they refer to, rather than being entered as part of the text. Through these links the further content of the objects may be accessed on demand. The lesson continues with a discussion of bijections and the possibility of using them to define the concept of one-to-one correspondence; we skip that page here and move on to the discussion of the equivalence of the two approaches to defining one-to-one correspondence.

```
One-to-One Correspondence: equivalence of two characterizations.

This discussion continues from the introduction of an alternative definition of
one-to-one correspondence THERE.

We have given three "definitions", of sorts, to the concept of one-to-one
correspondence between two classes, denoted by (f is 1-1 corr).
```

```
We gave a suggestive description of the concept informally, then gave a
different, purportedly equivalent, description in terms of the existence of
inverse functions, then a third explanation was bijection. In order make formal
reasoning about (f is 1-1 corr) possible we must somehow add new "axioms" about
this predicate. We could add some new primitive axioms that declare the
existence of this new predicate and some basic facts about it. But as usual, we
have elected to find a combination of concepts that we can informally understand
to be equivalent to one-to-one correspondence, and that we can already formally
reason about.

When there are several candidates for defining a new symbol, it is typical to
pick one as the principal definition, and demonstrate the equivalence of the
others. In our case, we chose to use

  DEF f is 1-1 corr == ∃ g:B→A. InvFuns(A;B;f;g)

as the principle definition, and prove

  THM* Bij(A;B;f)⟺f is 1-1 corr

as a theorem, although we could just as easily have chosen the reverse. The use
of this theorem will be pretty much as if it were a definition of
(f is 1-1 corr), but the "content" of the proof is essentially that being a
bijection is equivalent to the having an inverse.

Examination of the proof will reveal that it reduces to two lemmas expressing
the opposite directions of the equivalence, namely,

  THM* Bij(A;B;f) ⇒ ∃ g:B→A. InvFuns(A;B;f;g)

  THM* InvFuns(A;B;f;g) ⇒ Bij(A;B;f)

The proof of the first theorem shows how to construct an inverse of a function
given that it's a bijection. The second uses the inverse of a function to show
that it is a bijection.
```

When the user clicks on the theorem references, either the formal proof or an informal gloss of it is viewed. Here is a gloss of the last proof mentioned.

```
Here we gloss the proof of

THM* InvFuns(A;B;f;g) ⇒ Bij(A;B;f)
```

```
First note the key definitions:

  DEF InvFuns(A;B;f;g) == (∀ x:A. g(f(x)) = x) & ∀ y:B. f(g(y)) = y

  DEF Bij(A;B;f) == Inj(A;B;f) & Surj(A;B;f)

It turns out that each conjunct of the conclusion will follow from a conjunct of
the premise, namely,

  (∀ x:A. g(f(x)) = x) ⇒ Inj(A;B;f)

and

  (∀ y:B. f(g(y)) = y) ⇒ Surj(A;B;f).

To show Inj(A;B;f) is just to show that

  f(a1) = f(a2) ⇒ a1 = a2, which is so since

  f(a1) = f(a2) ⇒ g(f(a1)) = g(f(a2))

  which by assumption, rewrites to a1 = a2.

To show Surj(A;B;f) is just to show that

  ∀ b:B. ∃ a:A. f(a) = b

  which is witnessed by g(b) thus:

  ∀ b:B. f(g(b)) = b, which follows from our assumption.
```

The purpose of this gloss is informally imitate the main content of the formal proof, to which the reader may refer if further detail or resolution of ambiguity in the gloss is desired. Note that, as usual in Nuprl, the mathematical expressions are formal, rather than simply text, and so their definitions can be followed, and the way these abstract terms happen to be displayed is easily changed.

Now, let us turn to the formal proof. Proofs in Nuprl are trees of assertions, each of which is entailed by its children, or subgoals. There are two main modes for viewing Nuprl's formal proofs; one may either view the whole proof in a compact notation, or one may "walk" the proof tree, focusing on one inference step at a time.

There are a number of formats for these compressed representations of whole proofs; the one we shall use here shows the subproofs underneath the goal inferred from them, indenting them if there is more than one subproof.

```
⊢  InvFuns(A;B;f;g) ⇒ Bij(A;B;f)
1. A : Type
2. B : Type
3. f : A→B
4. g : B→A
5. InvFuns(A;B;f;g)
⊢  Bij(A;B;f)  by  Def of Bij(A;B;f)
|\
| ⊢  f is 1-1  by  Def of f is 1-1 and InvFuns(A;B;f;g)
| 5. ∀ x:A. g(f(x)) = x
| 6. a1 : A
| 7. a2 : A
| 8. f(a1) = f(a2)
| ⊢  a1 = a2  by  Apply g to both sides of 8
| 8. g(f(a1)) = g(f(a2))
| ⊢  a1 = a2  by  Rewrite 8 using 5
 \
  ⊢  Surj(A;B;f)  by  Def of Surj(A;B;f) and InvFuns(A;B;f;g)
  5. ∀ y:B. f(g(y)) = y
  6. b : B
  ⊢  ∃ a:A. f(a) = b  by  Witness: g(b) ....
  ⊢  f(g(b)) = b  by  BHyp 5 ....
```

In this proof format, any numbered assumptions that are the same for the child as for the parent are not shown again; this simplifies the reading.

Now, let us see what it's like to focus on the proof one inference at a time. The top inference looks like this:

```
* top
⊢  ∀ A,B:Type, f:A→B, g:B→A. InvFuns(A;B;f;g) ⇒ Bij(A;B;f)
BY UnivCD ...a
1* 1. A : Type
   2. B : Type
   3. f : A→B
   4. g : B→A
   5. InvFuns(A;B;f;g)
   ⊢  Bij(A;B;f)
```

The "* top" indicates that the proof is complete and that this is the top of the proof tree. The goal is followed by the tactic used for the inference, and the numbered list of subgoals automatically generated from the goal and the tactic. The inference step shown here is typical of the top of a proof; it break down the goal into the obvious assumptions and conclusion, and makes explicit the types of the variables used in the goal. If the user wishes to see the proof of a subgoal, clicking on it will show the inference step leading it from *its* subgoals.

```
* top 1
1. A : Type
2. B : Type
3. f : A⟶B
4. g : B⟶A
5. InvFuns(A;B;f;g)
⊢  Bij(A;B;f)
BY Def of Bij(A;B;f)
1* ⊢  Inj(A;B;f)
2* ⊢  Surj(A;B;f)
```

The "* top 1" indicates that the proof of this goal is complete, and also indicates its location within the tree of inference steps, namely, the first subgoal below the top of the proof. This step corresponds to splitting the conclusion into the two conjuncts after expanding its definition; the subgoals are numbered. The user would simply continue reading down the various branches of the proof until a goal with no subgoals is reached, or until he or she is satisfied.

It would take too much space here to walk the whole proof, so we will just show a couple other inference nodes. Here is the formal step which, in the gloss above, was expressed as trying to show that `f(a1) = f(a2) ⇒ a1 = a2` by first applying `g` to both sides of the premise.

```
* top 1 1 1
1. A : Type
2. B : Type
3. f : A⟶B
4. g : B⟶A
5. ∀ x:A. g(f(x)) = x
6. a1 : A
7. a2 : A
8. f(a1) = f(a2)
⊢  a1 = a2
BY Apply g to both sides of 8
1* 8. g(f(a1)) = g(f(a2))
   ⊢  a1 = a2
```

The subgoal differs from the goal simply by actually performing this application to the appropriate assumption. Notice that all the assumptions in force at each point in the proof are explicitly shown when walking the proof. Proceeding down to the proof of the new subgoal:

```
* top 1 1 1 1
1. A : Type
2. B : Type
3. f : A→B
4. g : B→A
5. ∀ x:A. g(f(x)) = x
6. a1 : A
7. a2 : A
8. g(f(a1)) = g(f(a2))
⊢  a1 = a2
BY Rewrite 8 using 5
```

This goal is proved, as was glossed above, simply by rewriting assumption 8 to `a1 = a2`, as is justified by assumption 5. This would generate a new subgoal in which the conclusion appears as an assumption, which the system recognizes as trivially justified, so there are no more subgoals below this point.

3 The Nuprl Computational Mathematics Medium

3.1 The Formal Language

This section gives an overview of features of Nuprl which bear on the proposal. We explain the role of a *typed language*, comment on the foundational theories Nuprl supports, describe the tactic-style logic engine and the special editors and library mechanism.

working mathematics The language of working mathematics is richly typed, starting with the numerical types: natural numbers $\mathbb{N}$, integers $\mathbb{Z}$, rationals $\mathbb{Q}$, reals $\mathbb{R}$, and complex numbers $\mathbb{C}$. *Cartesian products* consist of ordered pairs, as in $\mathbb{Z} \times \mathbb{N}$, pairs of integers and naturals. Functions are typed according to the types of their domain and range as in $\mathbb{Z} \rightarrow \mathbb{Q}$, the type of mappings of integers to rationals.

In its natural extent, this language includes non-numerical types such as *Propositions* and in metamathematics there are types of terms and formulas. The type *Proofs* is also sensible and useful, and various combinations produce common notions such as *propositional functions* over the integers, say $\mathbb{Z} \rightarrow$ Proposition.

types in programming and computer science

Types play a major role in most modern programming languages. We speak of the strong static typing of Java or polymorphic typing in ML. Classes and modules are also examples of types. The type system of a programming language such as ML bears a strong resemblance to the *mathematical types* discussed above.

Nuprl theories Nuprl 4 is a *generic system* in the sense that it can support any logic whose axioms are expressible in the type theory language. The present theory organization is a core type theory with optional axioms. But the students will not interact with the system at the level of axioms or inference rules. In CS 572, the Formal Methods course using Nuprl, we never had to mention a primitive rule. This is because users interact with the system at the level of very high-level derived rules, decision procedures, and proof building programs called *tactics*.

Tactics will be discussed in greater detail below, 2.2. But the basic idea is that tactics are programs that decompose proof goals into subgoals in such a way that if the subgoals are provable, then so is the goal. Indeed, users can generally see that if the subgoals are *true*, then so is the goal. So they operate at a *semantic level* with little or no concern for the basic rules. The important notion is the *collection of tactics*. We intend to base most of the interactive course material on a *dozen general tactics*. We will introduce them gradually and semantically. These play a large role in the course and in the system. For all practical purposes, the only logics we really support are those that the general tactics support.

3.2 Tactic-style Theorem Provers

Proofs in Nuprl are accomplished by repeatedly invoking *tactics* which refine goals to be proven into (usually simpler) subgoals. This kind of logic is called a Refinement Logic. [8, 9] Here is a description of a few of Nuprl's basic tactics. These are sufficient for completing most elementary proofs. A hypothesis is either a type declaration for a variable or it is an assumption. The hypotheses and conclusion of a sequent are collectively referred to as *clauses*.

- `D` i: Decompose the outermost connective in clause i. This tactic provides access to the introduction and elimination rules in Nuprl's logic. This tactic accepts optional arguments for, for example, instantiation of quantifiers.
- `Ind` i: Do induction on declaration i. The declaration must involve some inductively defined datatype: for example $\mathbb{N}$ or `T List` for some type `T`.
- `Assert` P: Split proof into two parts: one to prove P, the other assuming P to prove the original goal. This invokes Nuprl's *cut* rule.

- `Decide` P: Case split on whether proposition P is true or false.
- `Auto`: Do various kinds of automatic reasoning, including checking if conclusion is same as some assumption, certain decompositions, type checking, solving equalities and solving inequalities involving linear arithmetic expressions.
- `Backchain` as ls: Repeatedly back-chain using assumptions as and lemmas ls. This tactic is Nuprl's version of a Prolog-style resolution inference procedure.
- `ArithSimp` i: Put arithmetic expressions in clause i into a normal form.
- `RewriteWith` as ls i Rewrite clause i using assumptions as and lemmas ls.
- T_1 `THEN` T_2`,` T_1 `ORELSE` T_2`,` `Repeat` T: These are *tacticals* used for joining together tactics. `THEN` sequences tactics, `ORELSE` allows trying alternative tactics and `Repeat` repeatedly applies a tactic.

3.3 Built-in Functional Programming Language

One of the most interesting features of the Nuprl logic is that it supports computational reasoning as well as classical. In order to do that there must be a programming language implicit in the logic [9, 22]. In fact, the notations for computable functions make this language explicit. This enables us to deal with computational problems and issues for any mathematical concept.

The built-in programming language is essentially a subset of ML [34, 56] which can be used to *animate constructions and proofs*. For example, the proof that the `gcd`, say g, of two co-prime numbers a and b is a linear combination of a and y, say $g = a*n + b*m$ provides a function, f, to compute n and m from a and b. This computation can be done in-line as part of a lesson. We say that f is *extracted* from the proof. Students find it very appealing to see these *proofs come to life*.

The animation of proofs and the extraction of programs from them enables us to connect the mathematical notion of induction to the programming notion of recursion. We exploit this frequently. It illustrates the general principle that the formal connections between objects help teach the abstract links between concepts which appears to be a basic mechanism driving learning.

3.4 Library Structure and Use

The principal syntactic units of Nuprl are *terms* of an extensible, general purpose binding syntax. The data base the user works with is a *library* of objects, to which they may add. Other than proofs, all objects are terms that have been given names in the library, and proofs are special structures built from terms. Objects may be classified according to which of four roles they play: formally meaningful, informally meaningful, "utility", and display specification.

An operator *definition*, sometimes called an *abstraction*, is one kind of object, e.g., the definition of "`fog`." Proofs of theorems constitute another kind of formally meaningful object. A third kind of object is used to declare the basic inference rules, in terms of which all other more complex inferences are defined. Another kind of object defines procedures in the programming language ML for use in the proof tactics. Finally, the user may simply create objects to be used as data by tactics. These various kinds of objects constitute the *formally meaningful* part of the system about which precise semantic claims are made; ultimately, our claims about the truth of theorems depend upon them.

The *informally* meaningful objects are the various documents, such as lessons and notes, which typically refer to the formally meaningful components. Although these objects have no formal significance, they are essential to practical use of the library, providing the explanatory interface to the formal components. It is by means of such documents that one is guided through what could otherwise be a formal thicket whose significance is unclear; we call a document used as such a guide a *reading* of the library.

A third class of objects might be called *utilities*, which make using the system easier, such as menus for edit commands.

Finally, there are the objects that tell the system how to display terms. A fundamental design principle for Nuprl is this: *the formal content of an object is independent of the way it is to be displayed.*

This means that you never need commit to a particular notational decision since your changes to the display specification objects have automatic effect on viewing any objects of the system. For example:

- Whatever objects one person creates, others can read using whatever notations they prefer.
- When a person defines a new operator, he or she doesn't have to perfect the notation for it before using the operator. Notation may be incrementally improved.

- Notations can be redesigned for different audiences.

The fact that the formal content of an expression is largely independent of how it looks, is itself an important lesson about notation, as is the experience of improving one's notations.

4 Educational Aspects

In this section we look in more detail at some of the educational ideas behind this proposal.

4.1 Problems and Opportunities

problems with text The main application of Nuprl for this continuing study will be in creating the interactive lessons linked to the formal library. This unique material will solve a number of problems common to any technical subject, and in the future others might considerably extend the scope of what is proposed here. Let us consider the problems of reading mathematical texts.

From having read over 300 mathematical books and teaching mathematical subjects to over a thousand undergraduates I can document these universal problems that make reading mathematics difficult. First, many textbooks make mistakes because proofs are not checked; consistent use of definitions is not enforced, not all facts are clearly stated. Indeed, *every* discrete mathematics book that we studied made such errors. Second, there are the many *omissions*: justifications of steps are left out inadvertently at key places, steps themselves are left out, whole proofs are left out thinking sometimes that they are "just like the case proved" when they are not, cases in a case analysis are forgotten and so forth. Each omission can cost readers many hours of struggling. Third are the *location problems* or "access to resources," when it is hard to find a key definition, a lemma is cited by name but it is hard to find, some previous remark is cited but not located, sometimes a chain of definitions or theorems leads beyond the particular book. Fourth is the problem of *ambiguity*, a statement is made with several interpretations, an expression can be interpreted in different ways or some implicit parameter has been omitted.[6] *Motivation* for proof steps is missing, so a reader can imagine a simpler method to apply, but no explanation is given for this choice.

[6]For example, the Fundamental Theorem of Arithmetic is sometimes stated as "any natural number, other than 1, can be written in only one way as a product of primes." But students wonder about $3 \cdot 5 \cdot 2 \cdot 7 \cdot 3 \cdot$ versus $2 \cdot 3^2 \cdot 5 \cdot 7$.

These problems are more exasperating than a “boring lecture” because the educational process depends on a student digesting course material in peace and quiet, experimenting in a workspace that includes the assembled material of lecture notes, books and his or her own writing and computing.

problems with subgoaling One of the principles of learning studied using computerized tutors is that “solving a problem involves decomposing that problem into a set of goals and subgoals” [3, 4, 67]. Empirical studies have confirmed that interfaces to computerized tutoring systems which made explicit the goal structures were superior to those that did not [65]. The Nuprl proof editor was designed to decompose a goal into subgoals based on a problem solving method (a rule or a tactic) (see 1.4). This editor works across all domains formalized in Nuprl. It has proven to be especially effective in teaching induction.

4.2 Experience with Students Learning Discrete Mathematics and Logic

As we mentioned in the introduction, the emergence of the Web and Netscape changed our plans for deploying the courseware from what we initially proposed, and it offered new technology, CoNote, for evaluating its use. We are now incorporating this software into the project, which means that we have considerably more technology and more material than we promised, and our evaluation plans have been revised.

The first piece of new technology is software for automatically posting Nuprl libraries to the Web in a form nearly identified to their appearance inside the system. We used these libraries as supplemental material in two courses to see if they were usable. The results were positive and encouraging. We first used libraries on functions and automata with advanced students. Work with it led to improvements and the current Web lessons.

Next we used a large reference library on logic in an undergraduate logic course, Applied Logic. Our only feedback came from email questions, consulting (office hours) and assignments. We established that the Web material is a useful supplement to the textbook. Students had no difficulty relating the formal material to the informal textbook. In some cases, when the text provided only an informal algorithm on an informal definition, the students found the formal material especially helpful.

The lessons we have created are available on the web (see the url on the title page). The material will be used with sophomores in the fall and evaluated as we discuss in the next section.

4.3 Evaluation and Monitoring

monitoring patterns of use One of the possibilities opened up by the Web access to Nuprl libraries is that we can use CoNote to monitor how students use the interactive course material. We can find out which definitions are referred to most often, which proofs are read to great depth, which expressions must be fully disambiguated, which lemmas are most often viewed. These measures will give us a map of the difficulty of the cognitive terrain.

Our plan for the courseware offers a technique for evaluating the formalism. Since we plan to create ordinary hypertext lessons with links into and out of the formal text, we can monitor the student traffic into the formal text for various assignments.

formative evaluations Professor Jere Confrey [18] of the Cornell Education Department has agreed to meet with us as we plan systematic formative evaluations and assess the results.

We will use the Spring '98 offering of the Cornell discrete mathematics course as a control group for measuring the effectiveness of our course material to address the problems of omission, ambiguity, location of information and motivation discussed in 2. We will also use results on similar homework assignments to evaluate understanding of induction and logic. In general we will be looking to see if students use the material as we anticipate. Do they follow definition chains? Do they consult the formal proof when encountering obstacles to understanding the ordinary text? Do they often execute the algorithms associated with computational theorems in order to produce more examples of a result? Do they experiment by applying induction tactics to a variety of formulas to learn the structure of subgoals?

Integration of our courseware with the Cornell CoNote system [27] will enable students to ask questions about the course material interactively. We can reply interactively and add new formal or informal material as required.

4.4 Exploration and Collaboration

The existence of extensive libraries of formal mathematics in the system opens the possibility that interested students will explore on their own. For example, there is a proof of the FTA in the algebra library done for unique factorization domains. The account is a direct generalization of the results in the number theory library, so a student might learn about abstract algebra just by following leads.

We expect that students will contribute to the courseware not only by adding solutions to exercises, but from time to time, by creating new sub-libraries of material that interests them. Having this chance to create new formal mathematics or new algorithms might be an exciting outlet for budding scientists.

4.5 Criticisms

Of course, as with almost any significant enterprise, one can criticize various aspects of this proposal. Indeed, here are some criticisms one might expect along with a reply.

1. Some people who have never seen a system with a definition and display facility or who have only seen Lisp-like syntax for formal mathematics might complain that *formalism is too difficult to read.* But the examples presented here and in the appendix are taken directly from the system, and one can see for oneself how readable they are.

2. Since computer systems can be intrusive and consume a lot of time in programming courses, one might complain that *the computer will get in the way of content.* But the primary use of the software is for *reading* in the style of Netscape which are extremely convenient and can be explained to an elementary school student in minutes. This level of computer skill, "point-and-click," is a basic as reading. Furthermore, any advanced features of Nuprl which we find useful can also be presented in this style.

3. The formal aspect of the mathematics is one more concept to learn. It can be argued that *the added value is not worth the cost.* But the highest cost is for the *producer* of the material. We have already incurred much of that cost in our research since the mathematics is already formalized. We have shown that several problems are solved by the formalism (in 3.1). The project will attempt to assess the costs and benefits more carefully, especially the value of teaching logic as part of the formalization process rather than as an isolated topic.

4. There is some added cost to formalism, so someone might claim that the cost will make the material *inaccessible to average students.* There is no evidence to support this claim. Indeed, David Gries, a distinguished educator and co-author of the book *A Logical Approach to Discrete Mathematics* [36] has collected evidence to the contrary, showing that the average students benefit from the clarity and definiteness of a formal calculus. My own experience is similar as is experience with the more ambitious computerized tutors [3], and in a course teaching logic, some formalism must be taught.

5. The methods being investigated are only useful in mathematical courses with a rigorous foundation, and someone might claim that this is a *limited part of the curriculum* with

little impact below the second year in college. Of course, a great deal of mathematics is based on rigorous foundations. We can imagine these methods being helpful in algebra, geometry and calculus and eventually in physics, mechanical engineering, etc. Already there is a significant effort to deploy similar tools for geometry instruction in high school [3, 5]. These techniques have value both above and below the first and second year of college.

6. Finally, someone who is unwilling to support an experiment without knowing the outcome might say *The Cornell group does not have enough classroom experience with the tools* to guarantee success at the right level. But we are seeking the funds to acquire more experience and present the results of our work. We have offered the material in advanced courses with our own resources gaining a great deal of experience which informs this proposal. This led to the ideas presented here, a plan with modest aims and a large chance to succeed. The experience also led to changes in the system to make it more accessible. So we have already done the ground work.

In summary, we see that these criticisms are unwarranted for this project. The essential point is that we are promising to produce an exciting new product of wide value which will capitalize on a large NSF research investment to create added investment in education. We have a proven record of delivering excellent research, and we will deliver in education as well.

5 Work Plan

5.1 Background

PRL research group The work will be done by the proposer and researchers from the PRL research group, principally Robert Constable, Stuart Allen, Richard Eaton, and Lori Lorigo with help from others from time to time. We will be able to use a great deal of the material being created under various research grants and contracts, and many of the enhancements to Nuprl will apply in educational uses. We are asking only for funds to prepare course material and Web access to it.

5.2 Schedule and Milestones

Year 1 (January-December 1998)
The work this year would fall into two parts. In the first semester we will be testing,

improving, and collecting data about the success of our lessons on function and logic, counting and induction in the Cornell discrete mathematics course and from outside users (via CoNote on the Web). We will have advertized these lessons on the Web and at meetings such as FIE. We will be writing a report and improving the lessons as well as teaching the material and supporting others who are using the material remotely.

In the summer and second semester we will be using NuprlW to create fully interactive lessons as extensions of the Web lessons. We will test these as we did the lessons for discrete mathematics. We will also finish material for a senior level logic course by adding lessons on proof formats, specifically Tableau and Natural Deduction proofs.

Use of NuprlW will represent another major contribution of the Nuprl research program (funded by DARPA, ONR and NSF) to education.

Year 2 (January-December 1999)
The work will again divide into two parts. We will deploy the fully interactive lessons, both in discrete mathematics and in logic. The logic material will include a module we are calling the Gödel Project which will include an account of the Gödel incompleteness theorem.

In the second semester, we will either build lessons to enhance the automata library or the semantics of programming library depending on what subjects get developed further as part of our research program. At this point we will have a Courseware Library of nearly two dozen lessons that can support at least four courses in Mathematics and Computer Science.

The final phase of this grant would be to organize an increased group of instructors using our material. The details of this will depend on the exact composition of the users and their feedback to us.

Appendix

A An Exercise in Induction

To illustrate other possibilities for the courseware beyond its role as reference material, we present a sample exercise adapted from [52]. It would be used shortly after mathematical induction was introduced to the students, and after they have had some experience with the system. The exercise is to produce a simple proof by induction. The student begins by opening a window on the following informal document:

```
As a simple example involving induction, consider the "postage stamp"
problem. There will be a related exercise at the end.

Using just 3-cent and 5-cent stamps, it is possible to make up any
postage of at least 8 cents. This is not immediately obvious, since
we cannot simply imagine how to make such combinations for an
arbitrary postage. But there is a way to proceed from each such
postage the to the postage of one cent higher.

If we replace a single 5-cent stamp by two 3-cent stamps, or replace
three 3-cent stamps by two 5-cent stamps, this will raise the postage
by 1-cent. Of course, to guarantee we can make one of these
replacements, we have to have at least one 5-cent stamp or at least
three 3-cent stamps, and having a postage of 8 cents is
sufficient to guarantee this.

Now, if we were going to give a detailed proof about this, we would
probably not want to get involved in the details of stamp
collections, so as with any "word problem", we first reduce it to a
simpler one. Let us observe that you can make a postage of k-cents
just if

  ∃ x,y:ℕ. 3•x+5•y = k

Exercise1: Please, complete the proof of

           THM* ∀ y:{6...}. ∃ m,n:ℕ. 2•m+7•n = y.

           Notice that we have changed the stamp denominations. The
           menu VIEW→ stamps_proof_menu contains all the proof steps
           you will need. The only new inference rule you will find
           there is "Induction [clause#]" which we have only
           discussed; VIEW⟶ InductionNotes for review.
             You will need to use "CaseSplitOn <prop>" once to decide
           which branch of the stamp replacement procedure to take.
           NOTE: Please, put comments on the proof steps
           corresponding to the stamp replacement steps in the word
           problem.

Exercise2: Prove the same theorem, but use a significantly different
           case split.
```

The statement of the first exercise contains three hypertext links. The first points to the proof they are asked to complete, the second points to a short menu of inference tactics adequate for this problem, and the third points to a document containing introductory notes on induction which should support the in class discussion. When the student clicks on the theorem, a window is opened on the proof, which looks like this:

```
# top
⊢  ∀ y:{6...}. ∃ m,n:ℕ. 2•m+7•n = y
BY <refinement rule>
```

It consists only of the goal to be proved. In some other exercises, the proof supplied to the student will be partially proved; subgoals that are either to difficult or would distract from the point of the exercise would already be proved. This proof is pretty simple though.

Next, the student would probably open the menu of inference forms to be used with this problem by clicking on VIEW⟶ stamps_proof_menu. This would open a window that looks like this:

```
Induction⁺


Analyze [clause#]⁺
Witness <term>⁺
CaseSplitOn <prop>⁺


<tac> THEN <tac>⁺


Auto⁺
```

Each line of this menu consists of an "insert button" that causes the tactic clicked on to be inserted into the proof. At this stage, the student would already be familiar with using all the tactics listed in the menu except the one for induction. We have narrowed the solution space for this exercise so that the student will not get stuck, and so that the expected kind of proof will be produced.

The first step is to apply induction. If the student clicks on the menu entry `Induction`$^+$, the tactic will be inserted into the proof, and when the student hits return, the result will be:

```
# top
⊢  ∀ y:{6...}. ∃ m,n:ℕ. 2•m+7•n = y
BY Induction
1# .....basecase.....
   1. y : ℤ
   2. 6 = y
   ⊢  ∃ m,n:ℕ. 2•m+7•n = 6
2# 1. y : ℤ
   2. 6<y
   3. ∃ m,n:ℕ. 2•m+7•n = y-1  ...Ind Hyp
   ⊢  ∃ m,n:ℕ. 2•m+7•n = y
```

The result is two subgoals: the base case, and the induction case. Notice that the induction hypothesis is so-labeled.

The student may then click on either subgoal to descend into the proof. Let's say the base case is selected; this is what you'd see:

```
# top 1
.....basecase.....
1. y : ℤ
2. 6 = y
⊢  ∃ m,n:ℕ. 2•m+7•n = 6
BY <refinement rule>
```

The student should recognize that he or she can now specify witness for the conclusion. Going back to the tactic menu and clicking on `Witness <term>`$^+$ will simply insert `Witness <term>` into the proof, whereupon the student enters the expression for a witness into the slot marked `<term>`, say 3. Although it not necessary, the student could specify witness for both existential variables at once by chaining two `Witness` tactics with the `THEN` tactic from the menu. The result would be:

```
* top 1
.....basecase.....
1. y : ℤ
2. 6 = y
⊢  ∃ m,n:ℕ. 2•m+7•n = 6
BY Witness 3 THEN Witness 0
```

There are no more subgoals since the system recognizes the truth of the equation given these witnesses.

Clicking on the goal will take us back to the next inference step above, which now looks like

```
# top
⊢  ∀ y:{6...}. ∃ m,n:ℕ. 2•m+7•n = y
BY Induction
1* .....basecase.....
   1. y : ℤ
   2. 6 = y
   ⊢  ∃ m,n:ℕ. 2•m+7•n = 6
2# 1. y : ℤ
   2. 6<y
   3. ∃ m,n:ℕ. 2•m+7•n = y-1  ...Ind Hyp
   ⊢  ∃ m,n:ℕ. 2•m+7•n = y
```

The * indicates that that subproof is finished, leaving us to prove the other subgoal. Clicking on it will cause us to descend into that incomplete subproof and the only obviously (to the student) applicable tactic from the menu is to analyze the existential assumption of the inductive hypotheses.

```
# top 2
1. y : ℤ
2. 6<y
3. ∃ m,n:ℕ. 2•m+7•n = y-1  ...Ind Hyp
⊢  ∃ m,n:ℕ. 2•m+7•n = y
BY Analyze 3
1# 3. m : ℕ
   4. ∃ n:ℕ. 2•m+7•n = y-1
   ⊢
```

Then down into the new subgoal and analyze assumption 4 in the same way:

```
# top 2 1
1. y : ℤ
2. 6<y
3. m : ℕ
4. ∃ n:ℕ. 2•m+7•n = y-1
⊢  ∃ m,n:ℕ. 2•m+7•n = y
BY Analyze 4
1# 4. n : ℕ
   5. 2•m+7•n = y-1
   ⊢
```

In order to build witnesses for the conclusion from the variables of the hypotheses, we must know which way to do the stamp replacements suggested in the problem. Perhaps the student considers the replacing 3 2-cent stamps; then we need to know there are at least 3 of them to replace. The case splitting tactic from the menu is handy for this decision:

```
# top 2 1 1
1. y : ℤ
2. 6<y
3. m : ℕ
4. n : ℕ
5. 2•m+7•n = y-1
⊢  ∃ m,n:ℕ. 2•m+7•n = y
BY CaseSplitOn 3≤m
1# 6. 3≤m
   ⊢
2# 6. ¬3≤m
   ⊢
```

These two subgoals are easy to finish off:

```
* top 2 1 1 1
1. y : ℤ
2. 6<y
3. m : ℕ
4. n : ℕ
5. 2•m+7•n = y-1
6. 3≤m
⊢  ∃ m,n:ℕ. 2•m+7•n = y
BY Witness m-3 THEN Witness n+1
   ⟵ Replace 3 2-cent stamps by a 7-cent stamp.
```

and

```
* top 2 1 1 2
1. y : ℤ
2. 6<y
3. m : ℕ
4. n : ℕ
5. 2•m+7•n = y-1
6. ¬3≤m
⊢  ∃ m,n:ℕ. 2•m+7•n = y
BY Witness m+4 THEN Witness n-1
   ⟵ Replace one 7-cent stamp by 4 2-cent stamps.
```

The proof is then complete, as could be observed by going to the top of the proof and finding it marked with a *.

B Summary and Impact

We, of course, cannot predict all the problems that will arise, nor all the innovations that will result, but we can guarantee that we will generate a useful body of reference material, introductory lessons grounded in this material, collections of these lessons, and a proven mechanism for collecting feedback from readers of the material. Web access to this material will inform, if not *transform*, the teaching of various topics in college mathematics and computer science.

The research we are doing now combining Nuprl with a symbolic algebra system and our work in support of numerical computation have the potential to greatly broaden the scope and impact of our plans over the course of the next several years. So there is a chance that we can do even more than is scheduled here. The methodology proposed also has the potential to deepen our understanding of pedagogy. Our experience is that tactics are an effective way to teach technique. We can measure students progress as they learn to master the basic tactics. These combined with the basic tacticals (THEN, ORELSE, REPEAT) allow students to express methods of solving problems and will encourage them to reflect on skill acquisition.

As we make connections between more and more subjects, from logic to circuits and algebra, students will see that one major component of understanding is linking new knowledge tightly into a solid base of fundamental concepts.

References

[1] H. Abelson and G. J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, Cambridge, MA, 1985.

[2] A.V. Aho and J.D. Ullman. *Foundations of Computer Science*. Computer Science Press, New York, 1992.

[3] J. R. Anderson, F. S. Bellezza, and C. F. Boyle. The geometry tutor and skill acquisition. In J. R. Anderson, editor, *Rules of the Mind*, chapter 8. Erlbaum, Hillsdale, New Jersey, 1993.

[4] J. R. Anderson, C. F. Boyle, R. Farrell, and B. J. Reiser. Cognitive principles in the design of computer tutors. In P. Morris, editor, *Modeling Cognition*. Wiley, 1987.

[5] J. R. Anderson, C. F. Boyle, and G. Yost. The geometry tutor. *The Journal of Mathematical Behavior*, 5(20), 1986.

[6] John Barwise and John Etchemendy. *The Language of First-Order Logic*. Lecture Notes Number 23. Center for the Study of Language and Information, Stanford University, second edition, 1991.

[7] John Barwise and John Etchemendy. *Hyperproof*. Center for the Study of Language and Information, Stanford University, Stanford, California, 1994.

[8] J. L. Bates. *A Logic for Correct Program Development*. PhD thesis, Cornell University, 1979.

[9] J. L. Bates and Robert L. Constable. Proofs as programs. *ACM Trans. Program. Lang. and Syst.*, 7(1):53–71, 1985.

[10] Nancy Baxter, Ed Dubinsky, and Gary Levin. *Learning Discrete Mathematics with ISETL*. Springer-Verlag, New York, 1989.

[11] M. Beeson. The MathPert System, 1994. Personal Communication.

[12] Albert H. Beiler. *Recreations in the Theory of Numbers*. Dover Publications Inc., New York, 1964.

[13] R. S. Boyer and J. S. Moore. *A Computational Logic*. Academic Press, New York, 1979.

[14] J. S. Brown, A. Collins, and P. Duguid. Situated cognition and the culture of learning. *Educational Researcher*, 18(1):32–41, 1989.

[15] J. S. Bruner. *The Process of Education.* Harvard University Press, Cambridge, Massachusetts, 1960.

[16] A. Collins, J. S. Brown, and S. Newman. Cognitive apprenticeship: Teaching students the craft of reading, writing, and mathematics. In L. B. Resnick, editor, *Knowing, Learning, and Instruction: Essays in Honor of Robert Glaser*. Erlbaum, 1989.

[17] J. Confrey. A theory of intellectual development. In *For the Learning of Mathematics*, volume 14(3), pages 2–8, 1994.

[18] Jere Confrey. A review of the research on student conceptions in mathematics, science, and programming. In Courtney Cazden, editor, *Review of Research in Education*, chapter 1, pages 3–56. American Educational Research Association, 1990.

[19] Robert L. Constable. Type theory as a foundation for computer science. In *Theoretical Aspects of Computer Software, Int. Conf. TACS '91,* Sendai, Japan, Lecture Notes in Computer Science, Vol. 526, pages 226–243, 1991.

[20] Robert L. Constable. Experience using type theory as a foundation for computer science. In *Proceedings of the Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 266–279. LICS, June 1995.

[21] Robert L. Constable. Creating and evaluating interactive formal courseware for mathematics and computing. In Magdy F. Iskander, Mario J. Gonzalez, Gerald L. Engel, Craig K. Rushforth, Mark A. Yoder, Richard W. Grow, and Carl H. Durney, editors, *Frontiers in Education*, Salt Lake City, Utah, November 1996. IEEE.

[22] Robert L. Constable, Stuart F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, Douglas J. Howe, T.B. Knoblock, N.P. Mendler, P. Panangaden, James T. Sasaki, and Scott F. Smith. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall, NJ, 1986.

[23] Robert L. Constable and Douglas J. Howe. Implementing metamathematics as an approach to automatic theorem proving. In R.B. Banerji, editor, *Formal Techniques in Artificial Intelligence: A Source Book*, pages 45–76. Elsevier Science Publishers (North-Holland), 1990.

[24] Robert L. Constable, Paul B. Jackson, Pavel Naumov, and Juan Uribe. Constructively formalizing automata. In *Proof Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, Cambridge, 1997.

[25] Thierry Coquand and G. Huet. The Calculus of Constructions. *Information and Computation*, 76:95–120, 1988.

[26] Antonella Cupillari. *The Nuts and Bolts of Proofs.* Wadsworth Publishing Co., Belmont, California, 1989.

[27] James Davis and Daniel Huttenlocher. Shared annotations for cooperative learning. In *Proceedings of the ACM Conference on Computer Supported Cooperative Learning*, September 1995.

[28] John Dewey. *How We Think.* Prometheus Books, Buffalo, New York, 1991.

[29] R. Dyckhoff. Contraction-free sequent calculi for intuitionistic logic. In *The Journal of Symbolic Logic*, pages Vol. 57, Number 3, September 1992.

[30] Samuel Eilenberg. *Automata, Languages and Machines*, volume A. Academic Press, New York, 1974.

[31] R. Gagné and L. J. Briggs. *Principles of Instructional Design.* Holt, Rinehart and Winston, New York, 1974.

[32] J. H. Gallier. *Logic for Computer Science, Foundations of Automatic Theorem Proving.* Harper and Row, NY, 1986.

[33] D. R. Goldenson. Teaching introductory programming methods using structure editing: Some empirical results. In W. C. Ryan, editor, *Proceedings of the National Educational Computing Conference 1989*, pages 194–203, Eugene, Oregon, 1989. University of Oregon, International Council on Computers in Education.

[34] Michael Gordon, Robin Milner, and Christopher Wadsworth. *Edinburgh LCF: a mechanized logic of computation,* Lecture Notes in Computer Science, Vol. 78. Springer-Verlag, NY, 1979.

[35] N. Greenleaf. Bringing mathematics education into the algorithmic age. In Jr. J.P. Myers and M.J. O'Donnell, editors, *Constructivity in Comupter Science*, pages 199–217, New York, June 1991. Lecture Notes in Computer Science, Springer-Verlag.

[36] David Gries and Fred B. Schneider. *A Logical Approach to Discrete Math.* Springer-Verlag, New York, 1993.

[37] R. P. Grimaldi. *Discrete and Combinatorial Mathematics: An Applied Introduction.* Addison-Wesley, New York, 2nd edition, June 1989.

[38] Hans Hahn. The crisis in intuition. In *The World of Mathematics*, pages 1956–1976. Simon & Schuster, New York, 1956.

[39] Gila Hanna. Proofs that prove and proofs that explain. In G. Vergnaud, J. Rogalski, and M. Artigue, editors, *Proceedings of the International Group for the Psychology of Mathematics Education*, volume II, pages 45–51, Paris, 1989.

[40] Gila Hanna. Challenges to the importance of proof. In *For the Learning of mathematics*, volume 15. FLM Publishing Association, Vancouver, British Columbia, Canada, November 1995.

[41] D. Harel. *Algorithmics: The Spirit of Computing.* Addison-Wesley, Reading, MA, 1987.

[42] Guershon Harel and Ed Dubinsky, editors. *The Concept of Function, Aspects of Epistemology and Pedagogy*, volume 25. Mathematical Association of America, 1992.

[43] Leendert Helmink. *Tools for Proofs and Programs.* PhD thesis, Universiteit van Amsterdam, The Netherlands, 1992.

[44] John E. Hopcroft and Jeffrey D. Ullman. *Formal Languages and Their Relation to Automata.* Addison-Wesley, Reading, Massachusetts, 1969.

[45] Douglas J. Howe. Implementing number theory: An experiment with Nuprl. *8th International Conference on Automated Deduction,* Lecture Notes in Computer Science, Vol. 230, pages 404–415, July 1987.

[46] Paul B. Jackson. Exploring abstract algebra in constructive type theory. In A. Bundy, editor, *CADE12*, New York, June 1994. Springer-Verlag. LNAI.

[47] Paul B. Jackson. *Enhancing the Nuprl Proof Development System and Applying it to Computational Abstract Algebra.* PhD thesis, Cornell University, Ithaca, NY, January 1995.

[48] Paul B. Jackson. Developing a toolkit for floating-point hardware in the Nuprl proof development system. In *Advanced Research Workshop on Correct Hardware Design Methodologies*, pages 401–419, Organized by ESPRIT, Turin, Italy, June 1991.

[49] L.S. Jutting. *Checking Landau's 'Grundlagen' in the AUTOMATH System.* PhD thesis, Math Centre, Amsterdam, 1979.

[50] Edward Kasner and James R. Newman. Paradox lost and paradox regained. In *The World of Mathematics*, pages 1936–1955. Simon & Schuster, 1956.

[51] I. Lakatos. *Proofs and Refutations: The Logic of Mathematical Discovery.* Cambridge University Press, Cambridge, MA, 1976.

[52] C. L. Liu. *Elements of Discrete Mathematics.* McGraw-Hill, NY, 2nd edition, 1985.

[53] Zhaohui Luo. *Computation and Reasoning, A Type Theory for Computer Science.* Oxford University Press, New York, 1994.

[54] Wilbert J. McKeachie. *Teaching Tips: Strategies, Research and Theory for College and University Teachers.* D. C. Heath & Co., Toronto, 9th edition, 1994.

[55] Kenneth R. Meyer and Dieter S. Schmidt, editors. *Computer Aided Proofs in Analysis*, volume 28 of *The Institute for Mathematics and Its Applications.* Springer-Verlag, New York, 1991.

[56] R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML.* The MIT Press, 1991.

[57] A. Newell and H.A. Simon. *Human Problem Solving.* Prentice Hall, Englewood Cliffs, New Jersey, 1972.

[58] James R. Newman. *The World of Mathematics*, volume 3. Simon & Schuster, New York, 1956.

[59] O. Ore. *Invitation to Number Theory.* Mathematical Association of America, New York, 1967.

[60] S. Owre, J.M. Rushby, and N. Shankar. PVS: A prototype verification system. In In Deepak Kapur, editor, *Lecture Notes in Artificial Intelligence*, pages 748–752, Saratoga, NY, 1992. 11th International Conference on Automated Deduction (CADE), Springer-Verlag. No. 607.

[61] Erik Poll. *A Programming Logic Based on Type Theory.* PhD thesis, Technische Universiteit Eindhoven, 1994.

[62] Stephen Read, editor. *Computerised Logic Teaching Bulletin*, volume Vol. 2 No. 1. Bulletin of the Association for Computerised Logic Teaching, Scotland, March 1989.

[63] Thomas Reps and Tim Teitelbaum. *The Synthesizer Generator Reference Manual.* Springer-Verlag, New York, third edition, 1988.

[64] Kenneth H. Rosen. *Discrete Mathematics and Its Applications.* McGraw-Hill, New York, 2nd edition, 1991.

[65] R. Scheines and W. Sieg. An experimental comparison of alternative proof construction environments. Department of Philosophy CMU-PHIL-40, Carnegie Mellon University, Pittsburgh, Pennsylvania, August 1993.

[66] Alan H. Schoenfeld. *Mathematical Problem Solving*. Academic Press, Inc., New York, 1985.

[67] Herbert A. Simon. Problem solving and education. In D. Tuma and F. Reif, editors, *Problem Solving and Education: Issues in Teaching and Research*. Erlbaum, Hillsdale, New Jersey, 1980.

[68] D. Sleeman and J. S. Brown, editors. *Intelligent Tutoring Systems*. Academic Press, London, 1982.

[69] Daniel Solow. *How to Read and Do Proofs: An Introduction to Mathematical Thought Process*. John Wiley & Sons, New York, 1982.

[70] E. Soloway and K. Ehrlich. Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, SE-10(5):595–609, 1984.

[71] Donald F. Stanat and David F. McAllister. *Discrete Mathematics in Computer Science*. Prentice-Hall, Englewood Cliffs, New Jersey, 1977.

[72] Patrick Suppes. *University-Level Computer-Assisted Instruction at Stanford: 1968–1980*, volume IMSSS. Stanford University, Stanford, California, 1981.

[73] David Tall, editor. *Advanced Mathematicial Thinking*. Kluwer Academic Publishers, 1991.

[74] William P. Thurston. On proof and progress in mathematics. In *For the Learning of Mathematics*, volume 15. FLM Publishing Association, February 1995. Vancouver, British Columbia, Canada.

[75] B. A. Trakhtenbrot. *Algorithms and Automatic Computing Machines*. D. C. Heath and Company, Lexington, Massachusetts, 1963.

[76] A. Trybulec and P. Rudnicki. Using Mizar in computer aided instruction of mathematics. To appear, 1993.

[77] Daniel J. Valleman. *How to Prove It: A Structured Approach*. Cambridge University Press, New York, 1994.

[78] R. Wertheimer. The geometry proof tutor: An ”intelligent” computer-based tutor in the classroom. *Mathematics Teacher*, pages 308–313, 1990.

[79] A. N. Whitehead. *The Aims of Education*. MacMillan, New York, 1929.

[80] A.N. Whitehead and B. Russell. *Principia Mathematica*, volume 1, 2, 3. Cambridge University Press, 2nd edition, 1925-27.

[81] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison-Wesley, 1988.