Implementing Mathematics with the Nuprl Proof Development System

R.L. Constable
S.F. Allen
H.M. Bromley
W.R. Cleaveland
J.F. Cremer
R.W. Harper
D.J. Howe
T.B. Knoblock
N.P. Mendler
P. Panangaden
J.T. Sasaki
S.F. Smith

Implementing Mathematics

with

the Nuprl Proof Development System

R.L. Constable
S.F. Allen
H.M. Bromley
W.R. Cleaveland
J.F. Cremer
R.W. Harper
D.J. Howe
T.B. Knoblock
N.P. Mendler
P. Panangaden
J.T. Sasaki
S.F. Smith

Computer Science Department
Cornell University
Ithaca, NY

Prentice-Hall, Inc., Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

Implementing mathematics with the Nuprl proof development system.

Bibliography: p.
Includes index.
1. Automatic theorem proving. 2. Nuprl (Computer system) 3. Mathematics—Data processing. I. Constable, R. L. (Robert L.) II. Cornell University. Dept. of Computer Science.
QA76.9.A96I47 1986 511.3 86-8197
ISBN 0-13-451832-2

Editorial/production supervision: LISA SCHULZ
Interior design: THE PRL GROUP
Cover design: EDSAL ENTERPRISES
Manufacturing buyer: GORDON OSBOURNE

This research supported in part by the National Science Foundation under grant DCR 83–03327.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-451832-2 025

Prentice-Hall International (UK) Limited, *London*
Prentice-Hall of Australia Pty. Limited, *Sydney*
Prentice-Hall Canada Inc., *Toronto*
Prentice-Hall Hispanoamericana, S.A., *Mexico*
Prentice-Hall of India Private Limited, *New Delhi*
Prentice-Hall of Japan, Inc., *Tokyo*
Prentice-Hall of Southeast Asia Pte. Ltd., *Singapore*
Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*
Whitehall Books Limited, *Wellington, New Zealand*

Preface

We hope to accomplish four things by writing this book. Our first goal is to offer a tutorial on the new mathematical ideas which underlie our research. In doing so we have tried to provide several entry points into the material, even at the cost of considerable redundancy. We hope that many of the ideas will be accessible to a well–trained undergraduate with a good background in mathematics and computer science. Second, parts of this book should serve as a manual for users of the Nuprl system (pronounced "new pearl"). As the system has grown so has the demand, both here at Cornell and at other institutions, for better documentation. We have tried to collect here all material relevant to the operation of the system. Third, we give an overview of our project for those interested in applications of the results and for those inclined to basic research in the area. Finally, we present new research which has arisen as we have worked on the Nuprl system. This system embodies contributions to the foundations of computer science and semantics, to automated reasoning and to system design, and it has shown promise as an intelligent system.

Authoring this book was a collective task; many individual efforts found their ways into these pages. Most chapters have several authors. That we were able to proceed in this fashion owes to the fact that we have assembled at Cornell a unique group of computer scientists who had worked for more than two years on the project.

We would like to acknowledge warmly the special contributions of Joseph L. Bates to this enterprise. Joe has been a chief architect of this system and was a major force in creating the PRL project, from which the system emerged. This manuscript is replete with Joe's ideas. We also want to thank Evan Steeg, who joined the project as an undergraduate and has since contributed significantly to our efforts, especially in the area of writing tactics. Tim Griffin's implementation of the rules for recursive types and partial functions is greatly appreciated as is his contribution of a new interface to the evaluator. All of our efforts are built on the contributions of Fran Corrado, our only full–time programmer. We also thank Christoph Kreitz, who has used the system extensively and provided us with insightful reports on its strengths and weaknesses. We also appreciate the constructive criticism of James Hook, who spent many hours reading and discussing early drafts. We thank Ryan Stansifer, Hal Perkins, and Aleta Ricciardi for proof–reading. Finally, we thank Donette Isenbarger, Michele Fish and Dawn Hall for their conscientious assistance in preparing the manuscript.

We appreciate the help we have received from the National Science Foundation through a series of grants supporting the Nuprl project, the Cor-

nell Computer Science Department computing facility, and various pieces of equipment needed for this research (MCS80-03349, MCS81-04018, DCR80-03327, DCR81-05763, and DRC84-06052). The NSF has also supported Mr. Cleaveland and Mr. Smith with fellowships. We also appreciate the generous support of IBM, whose fellowships have supported Mr. Mendler. Finally, we acknowledge the support of Cornell University and especially our department for providing fellowships, matching funds, and the environment to make this work possible.

This is the first version of the book. We consider it to be a preliminary effort and welcome timely comments that will help us improve the presentation. The book describes version 1.0 of Nuprl.

Robert L. Constable
for the PRL Group
Ithaca, NY

Contents

Part III: Advanced

Chapter 1

Overview

1.1 Brief Description of Nuprl

Problem solving is a significant part of science and mathematics and is the most intellectually significant part of programming. Solving a problem involves understanding the problem, analyzing it, exploring possible solutions, writing notes about intermediate results, reading about relevant methods, checking results, and eventually assembling a solution. Nuprl is a computer system which provides assistance with this activity. It supports the interactive creation of proofs, formulas, and terms in a formal theory of mathematics; with it one can express concepts associated with definitions, theorems, theories, books and libraries. Moreover, the theory is sensitive to the computational meaning of terms, assertions and proofs, and the system can carry out the actions used to define that computational meaning. Thus Nuprl includes a programming language, but in a broader sense it is a system for implementing mathematics.

1.2 Highlights of Nuprl

One of the salient features of Nuprl is that the logic and the system take account of the computational meaning of assertions and proofs. For instance, given a constructive existence proof the system can use the computational information in the proof to build a representation of the object which demonstrates the truth of the assertion. Such proofs can thus be used to provide data for further computation or display. Moreover, a proof that for any object x of type A we can build an object y of type B satisfying relation $R(x, y)$ implicitly defines a computable function f from A to B. The system can build f from the proof, and it can evaluate f on inputs of type A. For example, given any mapping f of a nonempty set A onto a finite set B of

smaller but nonzero cardinality, one can say that there will be two points of A mapped to the same point of B. From a proof of this statement the system can extract a function which given specific A, B and f produces two points of A mapped to the same point of B. This function expresses the computational content of the theorem and can be evaluated.

As a computer system Nuprl supports an interactive environment for text editing, proof generation and function evaluation. The interaction is oriented around a computer terminal with a screen and a mouse, for our intention is to provide a medium for doing mathematics different from that provided by paper and blackboard. Eventually such a medium may support a variety of input devices and may provide communication with other users and systems; the essential point, however, is that this new medium is active, whereas paper, for example, is not. This enables the interactive style of proof checking that characterizes Nuprl; in this system it is impossible to develop an incorrect proof.

Nuprl also possesses some of the characteristics of an intelligent computer system in that it provides its users with a facility for writing proof–generating programs in a metalanguage, ML. The implementation of the logic codes into Nuprl certain primitive mathematical knowledge in the form of rules for generating proofs and in the form of certain defined types in ML. As people use Nuprl they create libraries of mathematical facts, definitions and theorems; they can also create libraries of ML programs which use these results and other ML programs to generate proofs of theorems. In a very real sense, as Nuprl is used its capacity for providing help in proving theorems increases. By virtue of this property, Nuprl possesses aspects of an intelligent system.

The system design exhibits several key characteristics. The style of the logic is based on the stepwise refinement paradigm for problem solving in that the system encourages the user to work backward from goals to subgoals until one reaches what is known. Indeed, the system derives its name, Proof Refinement Logic,[1] from this method of presentation. The logic has a constructive semantics in that the meaning of propositions is given by rules of use and in terms of computation. We discuss these features in more detail later.

In a larger sense the Nuprl system serves as a tool for experimenting with ways of applying computer power to solving problems and to generating exact explanations of solutions, especially in the realm of computational mathematics. Because the difficult part of computer programming is precisely in problem solving and in explaining algorithmic solutions, we think that a system of this kind will eventually have a lasting impact on our ability to produce reliable and understandable programs.

[1]Nuprl is version "nu" of our Proof Refinement Logics. Earlier versions were "micro" and "lambda" [Nuprl Staff 83].

1.3 Motivations

In high school algebra solutions to most problems can be checked by simple computation; for instance, one can easily verify by substitution and reduction that 17 is a root of $x^3 - 16x^2 - 19x + 34$. Those who are pleased with the certainty of such solutions may also hope to find ways of checking solutions to more abstract problems (such as showing that $\sqrt{2}^{\sqrt{2}}$ is irrational) computationally. The idea that computers can check proofs is a step toward achieving this ambition [McCarthy 62,deBruijn 80], and there are several interesting implementations of this idea [Suppes 81,Weyhrauch 80,Constable, Johnson, & Eichenlaub 82,Gordon, Milner, & Wadsworth 79]. The computer system described here represents another approach to this problem.

Someone who has struggled for hours or perhaps days to untangle the details of a complex mathematical argument will understand the dream of building a computer system which helps *fill in the details* and keeps track of the proof obligations that must be met at the critical points of an argument. There are computer systems which do this, and it is possible in principle to solve problems in a system of this kind which are beyond the patience and capability of an unaided human being, as K. Mulmuley has already demonstrated in LCF [Mulmuley 84]. Nuprl uses some of the mechanisms of LCF which make this possible, namely the metalanguage ML in which the user writes programs to provide help with the details.

People who have spent hours or days getting a mathematical construction such as Gauss' construction of a regular 17–gon exactly right will know the desire to watch this mathematical construction "come to life." Nuprl was built to meet such aspirations. Every construction described in Nuprl's mathematics can, in principle, be executed mechanically. In particular, Nuprl can execute functions and thus serve as a programming language in the usual sense.

Anyone who has tried to write a mathematical paper or system consisting of several interacting theorems and constructions will appreciate having a uniform notation for discourse and a facility for creating an encyclopedia of results in this notation. The Bourbaki effort [Bourbaki 68] manifests such aspirations on a grand scale, and Nuprl is a step toward realizing this goal. It supports a particular mathematical theory, *constructive type theory*, whose primitive concepts can serve as building blocks for nearly any mathematical concept. Thus Nuprl provides a uniform language for expressing mathematics. This is a characteristic that distinguishes Nuprl from most other proof–checking or theorem–generating systems.

Anyone who has written a heuristic procedure such as one for playing games or finding proofs and who has seen it do more than was expected will understand the dream of using a computer system which can provide unexpected help in proving theorems. It is easy to extrapolate to dreams of machines offering critical help in solving real problems. Substantial effort has

been put into "theorem–proving" programs which attempt to provide just such help, usually in a specific domain of mathematics. While Nuprl is not a theorem–proving program in the usual sense, it is a tool for exploring this kind of heuristic programming. Users may express a variety of procedures which search for proofs or attempt to fill in details of a proof. The system has already provided interesting experiences of unexpected behavior by finishing proofs "on its own."

People who have experienced the new electronic medium created by computers may imagine how mathematics will be conducted in it. We see electronic mail and electronic text editing. We see systems like the Cornell Program Synthesizer [Teitelbaum & Reps 81] that help users specify objects such as programs directly in terms of their structure, leaving the system to generate the textual description. We can imagine these ideas being applied to the creation of mathematical objects such as functions and proofs and their textual descriptions. In this case, because mathematical structure can be extremely complex, one can imagine the computer providing considerable assistance in producing objects as well as help in displaying the text and giving the reader access to the underlying structure in various ways. Nuprl offers such capabilities, the most striking among which are the help the system provides in writing formal text (see chapter 3) and in reading and generating highly structured objects such as proofs and function terms. As with all aspects of Nuprl there is much to be done to achieve the goals that motivated the design, but in every case the system as it stands makes a contribution. We hope it will show the way to building better systems of this kind.

1.4 The Nuprl Logical Language

An algorithm to add two integers can be expressed in Nuprl as $\backslash x.\backslash y.x + y$. In general, if $b(x)$ is an expression such as $x * 2$ or $\backslash y.x + y$ in the variable x, then $\backslash x.b(x)$ is the function of one argument which on input a computes the value $b(a)$, where $b(a)$ stands for the expression b with each occurrence of x replaced by a. Application of the function is expressed by $(\backslash x.b(x))(a) = b(a)$; for example, $(\backslash x.x * 2)(2) = 2 * 2 = 4$. We speak of $\backslash x.b(x)$ as the *abstraction* of $b(x)$ with respect to x. This notation is essentially the same as Church's lambda notation, which is systematized in the *lambda calculus* [Church 51]. The form $\backslash x.b(x)$ is a visual approximation to $\lambda x.b(x)$ of the lambda calculus. Our notation is also very close to that used in ML [Gordon, Milner, & Wadsworth 79] except that type information about the variable can be included in the ML form. This marks a significant departure from the Algol–like languages as well, for in these languages the type information is given with the parameters of a function. For example, one would write **function**(x:**int**)**int** in the heading of an Algol function

definition to show that the function maps integers to integers.

In Nuprl, as in the work of H.B. Curry [Curry, Feys, & Craig 58,Curry, Hindley, & Seldin 72], a type discipline is imposed on algorithms to describe their properties. Thus, when we say that $\backslash x.x+1$ is a function from integers to integers, we are saying that when an integer n is supplied as an argument to this algorithm, then $n+1$ will denote a specific integer value. In general, meaning is given to a function term $\backslash x.b(x)$ by saying that it has type $A \rightarrow B$ for some type A called the *domain* and some type B called the *range*. The type $A \rightarrow B$ denotes functions which on input a from A produce a value in B. The fact that the functions always return a value is sometimes emphasized by calling them *total functions*.

The type structure hierarchy of Nuprl resembles that of *Principia Mathematica*, the ancestor of all type theories. The hierarchy manifests itself in an unbounded cumulative hierarchy of universes, U_1, U_2, ..., where by cumulative hierarchy we mean that U_i is in U_{i+1} and that every element of U_i is also an element of U_{i+1}. Universes are themselves types,[2] and every type occurs in a universe. In fact, A is a type if and only if it belongs to a universe. Conversely all the elements of a universe are types.

It is pedagogically helpful to think of the other Nuprl types in five stages of decreasing familiarity. The most familiar types and type constructors are similar to those found in typed programming languages such as Algol 68, ML or Pascal. These include the atomic types `int`, `atom` and `void` along with the three basic constructors: cartesian product, disjoint union and function space.[3] If A and B are types, then so is their cartesian product, $A\#B$, their disjoint union, $A|B$, and the functions from A to B, $A\text{->}B$.

The second stage of understanding includes the dependent function and dependent product constructors, which are written as $x{:}A\text{->}B$ and $x{:}A\#B$, respectively, in Nuprl. The dependent function space is used in AUTOMATH and the dependent product was suggested by logicians studying the correspondence between propositions and types; see Scott [Scott 70] and Howard [Howard 80]. These types will be explained in detail later, but the intuitive idea is simple. In a dependent function space represented by $x{:}A\text{->}B$, the range type, B, is indexed by the domain type, A. This is exactly like the indexed product of a family of sets in set theory, which is usually written as $(\Pi x \in A)B(x)$ (see [Bourbaki 68]). In the dependent product represented by $x{:}A\#B$, the type of the second member of a pair can depend on the value of the first. This is exactly the indexed disjoint sum of set theory, which is usually written as $(\Sigma x \in A)B(x)$ (see [Bourbaki 68]).

[2]The concept of a universe in this role, to organize the hierarchy of types, is suggested in [Artin, Grothendieck, & Verdier 72] and was used by Martin-Löf [Martin-Löf 73]. This is a means of achieving a *predicative* type structure as opposed to an *impredicative* one as in Girard [Girard 71] or Reynolds [Reynolds 83].

[3]In the discussion which follows we will use **typewriter** font to signify actual Nuprl text and *mathematical* font for metavariables.

The third stage of understanding includes the quotient and set types introduced in [Constable 85]. The set type is written $\{x{:}A \mid B\}$ and allows Nuprl to express the notions of constructive set and of partial function. The quotient type allows Nuprl to capture the idea of equivalence class used extensively in algebra to build new objects.

The fourth stage includes propositions considered as types. The atomic types of this form are written a = b `in` A and express the proposition that a is equal to b in type A. A special case of this is a = a `in` A, written also as a `in` A. These types embody the propositions–as–types principle discovered by H. B. Curry [Curry, Feys, & Craig 58], W. Howard [Howard 80], N. G. deBruijn [deBruijn 80] and H. Lauchli [Lauchli 70] and used in AUTOMATH [deBruijn 80,Jutting 79]. This principle is also central to P. Martin-Löf's Intuitionistic type theories [Martin-Löf 82,Martin-Löf 73].

The fifth stage includes the recursive types and the type of partial functions as presented by Constable and N. P. Mendler [Constable & Mendler 85]. These are discussed in chapter 12, but they have not been completely implemented so we do not use them in examples taken directly from the computer screen. These are the only concepts in this book not taken directly from the 1984 version of the system.

The logical language is interesting on its own. It is built around tested logical notions from H. Curry, A. Church, N. deBruijn, B. Russell, D. Scott, E. Bishop, P. Martin-Löf and ourselves among others and as such forms part of a well–known and thoroughly studied tradition in logic and philosophy. However, this tradition has not until now entered the realm of usable formal systems and automated reasoning in the same way as the first–order predicate calculus. Thus the design and construction of this logic is a major part of Nuprl, and becoming familiar with it will be a major step for our readers. We recommend in addition to this account the forthcoming book of B. Nordström, K. Petersson and J. Smith, the paper "Constructive Validity" [Scott 70], the paper "Constructive Mathematics and Computer Programming" [Martin-Löf 82] and "Constructive Mathematics as a Programming Logic" [Constable 85]. An extensive guide to the literature appears in the references. We hope that this account will be sufficient to allow readers to use the system, which in the end may be its own best tutor.

1.5 Components of the Nuprl System

General

The Nuprl system has six major components: a window manager, a text editor, a proof editor, a library module, a command module and a function evaluator. Interaction with Nuprl takes place in the context of three languages—the command language, which is extremely simple, the object

language, which is the mathematical language of the system, and the meta-language, which is a programming language with data types referring to the object language. The command language is used to initiate editing, use the library and initiate executions in the object and metalanguages, to name a few of its functions. The object language is a constructive theory of types. The metalanguage is the programming language ML from the Edinburgh LCF system modified to suit Nuprl.

The window manager provides the interface for the interactive creation of certain kinds of linguistic objects called *definitions*, *theorems*, *proofs* and *libraries* on a terminal screen. Windows offer views of objects; using these views the user can navigate through the object, stopping to modify it, to copy parts of it to other windows or to insert parts into it from other windows, etc. Three special windows provide for editing theorems, viewing the library and entering commands.

Definitions

The definition facility allows one to develop new notations in the form of templates which can be invoked when entering text. This feature provides a flexible way to introduce new notation. For instance, we might have defined a function, `abs(x)`, which computes the absolute value of a number, but we might like to display it in the text as `|x|`. This can be accomplished by defining a template. The format is `|<x:number>| == abs(<x>)`, where the left–hand side of == is the template and the right–hand side is the value (the angle brackets are used to designate the parameter).

Proof Editing

The proof–editing facility supports top–down construction of proofs. Goals are written as *sequents*; they have the form $x_1 : H_1, \ldots, x_n : H_n \gg G$, where the H_i are the hypotheses and G is the conclusion. To prove a goal, the user selects a rule for making progress toward achieving this goal, and the system responds with a list of subgoals. For example, if the goal is prove $A\&B$ and the rule selected is "and introduction," the system generates the following subgoals.

1. prove A
2. prove B

Proofs can be thought of as finite trees, where each node corresponds to the application of a logical rule. A proof can be read to the desired level of detail by descending into subtrees whose structure is interesting and passing over others.

Evaluation

The evaluation mechanism allows us to regard Nuprl as a functional programming language. For example, the multiplication function in Nuprl notation is `\x.\y.(x*y)`, and one can evaluate `(\x.\y.(x*y))(2)(3)` to obtain the value 6. Of course, one can define much more complex data such as infinite precision real numbers and functions on them such as multiplication (see section 11.6). In this case we might also evaluate the form `(\x.\y.mult(x,y))(e)(pi)(50)`, which might print the first 50 digits of the infinite precision multiplication of the transcendental numbers e and π.

The evaluation mechanism can also use a special form, `term_of(`t`)`, where t is a theorem. The `term_of` operation extracts the constructive meaning of a theorem. Thus if we have proved a theorem named t of the form *for all x of type A there is a y of type B such that $R(x, y)$* then `term_of(`t`)` extracts a function mapping any a in A to a pair consisting of an element from B, say b, and a proof, say p, of $R(a, b)$. By selecting the first element of this pair we can build a function f from A to B such that $R(x, f(x))$ for all x in A. The system also provides a mechanism for naming and storing executable terms in the library.

1.6 Programming Modes

The `term_of` operation enables two new modes of programming in Nuprl in addition to the conventional mode of writing function terms. To program in the first new mode one writes *proof outlines*, p, which contain the computational information necessary for `term_of(`p`)` to be executable. The second mode is a refinement of the first in which complete proofs, cp, are supplied as arguments to `term_of`. In this case one knows that the program meets its specification, so this mode might be called "verified programming".

1.7 Physical Characteristics

The system described here is written in about 60,000 lines of Lisp. It runs in Zetalisp on Symbolics Lisp Machines and in Franz Lisp under Unix 4.2BSD. There are also several thousand lines of ML programs included in the basic system. We have been using it since June 1984 principally to test the ideas behind its design but also to begin accumulating a small library of formal algorithmic mathematics. It can be used as a programming environment, and to a limited extent it has seen such service. But unlike its simpler predecessor, Lambda–prl [Nuprl Staff 83], it has not been provided with a compiler nor an optimizer, so it can be very inefficient. We hope that in due course there will be facilities to make it run acceptably fast. The system is in fact growing, but the major thrust over the next year is to substantially

enhance its deductive power and its user–generated knowledge in the form of libraries of definitions, theorems and proof methods (see section 1.9 for some of our detailed plans).

1.8 The "Feel" of the System

Nuprl is an example of an entity which is more than the sum of its parts. It is more than a proof checker or proof–generating editor. It is more than an evaluator for a rich class of functional expressions and more than a system for writing heuristic procedures to find proofs in a specific foundational theory. The integration of these parts creates a new kind of system. One can sense new possibilities arising from this combination both in the system as it exists now and in its potential for growth.

We find that Nuprl as it runs today serves not only as a new tool for writing programs and program specifications but also as a tool for writing nearly any kind of mathematics. Working in the Nuprl environment results in a distinctive style of mathematics—a readable yet formal algorithmic style. The style in turn suggests new mathematical substance such as our treatment of recursive types and partial functions [Constable & Mendler 85]. Nuprl also offers a coherent way to organize and teach a collection of concepts that are important in computer science.

As we extrapolate the course of Nuprl's development we see the emergence of a new kind of "intelligent" system. The mechanisms for the accumulation of mathematical knowledge in the form of definitions, theorems and proof techniques are already in place, and as more of this knowledge is accumulated the system exhibits a more widely usable brand of formal mathematics. Furthermore, since the tactic mechanism gives the system access to the contents of its libraries, one can envision altering the system so that it generates and stores information about itself. In this sense Nuprl is an embryonic intelligent system.

1.9 This Document

Scope

More than just a user's manual for a specific system, this book serves as an introduction to a very expressive foundational theory for mathematics and computer science, a theory which brings together many diverse ideas in modern computing theory. The book is also an introduction to formal logic in general and to constructive logic in particular, and it introduces new methods of program development and verification.

This book also describes a computer system for doing mathematics. The system provides a medium distinct from the traditional paper and black-