Chapter 1

Introduction

The NUPRL proof development system is a framework for the development of formalized mathematical knowledge as well as for the synthesis, verification, and optimization of software. It is based on a significant extension of Martin-Löf's intuitionistic Type Theory [ML84], which includes formalizations of the fundamental concepts of mathematics, data types, and programming. The system itself supports interactive and tactic-based reasoning, decision procedures, evaluation of programs, language extensions through user-defined concepts, and an extendable library of verified knowledge from various domains.

Since its first release in 1984 [CAB^{+}86], the system has been undergone several significant modifications to meet the growing demands for formal knowledge and tools in programming and mathematics, the most recent being a complete redesign of its architecture, discussed in [ACE^{+}00]. The NUPRL 5 system, which is documented in this manual, features an open, distributed architecture that integrates all its key subsystems as independent components and uses a flexible knowledge base as its central component.

1.1 The NUPRL 5 Architecture

Figure 1.1 illustrates the architecture of NUPRL 5. The system is organized as a collection of communicating processes that are centered around a common knowledge base, called the *library*. The

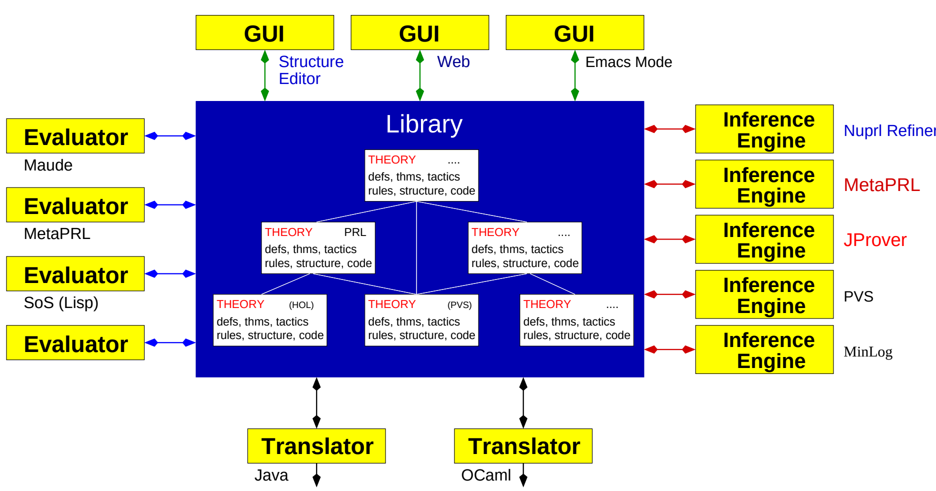

Figure 1.1: NUPRL 5 distributed open architecture

library contains definitions, theorems, inference rules, meta-level code (e.g. tactics), and structure objects that can be used to provide a modular structure for the library's contents. Inference engines (*refiners*), user interfaces (*editors*), rewrite engines (*evaluators*), and *translators* are started as independent processes that can connect to the library at any time.

The library can communicate with arbitrarily many other processes. This allows the user to connect several refiners and evaluators simultaneously, e.g. the NUPRL and MetaPRL [Met] refiners, proof systems like HOL [GM93] or PVS [ORR+96], first-order provers like JProver [SLKN01], Otter [WWM+90], EQP [McC97], or Setheo [LSBB92], proof-based program generators like MinLog [BBS+98], rewrite engines like Maude [CDE+99a], computer algebra systems like Mathematica [Wol88] or Maple [Map], decision procedures [NO79, Sho84, SVC], and model checkers [McM93, Dil96, Hol97], and even to make them cooperate. It is also possible to run different refiners in parallel on the same proof goal or several instances of the same refiner on different proof goals.

Providing several editors enables several users to work in parallel on the same formal theory while using their favorite interface. At the same time external users can access the system through the Web without having to start the whole system themselves.

Translators between the formal knowledge stored in the library and, for instance, programming languages like Java or Ocaml [Kre97, KHH98, Kre03] allow the formal reasoning tools to supplement real-world software from various domains.

NUPRL 5 is highly configurable. In its current standard configuration, which is described in this manual, the system essentially provides an extended functionality of the NUPRL 4 system [Jac94]. It consists of the library, the NUPRL 5 editor, and the NUPRL 5 refiner.

The Knowledge Base

The knowledge base is based on a transaction model for entering and modifying objects. All changes to objects, e.g. the effects of editor commands or inference steps, are immediately committed to the persistent library. The knowledge base also provides the option to undo changes, redo transactions, or to have several processes view or work on the same object – essentially following the same protocols as databases. However, changes do not overwrite an object but instead create a new version. The previous version is preserved until it is explicitly destroyed in a garbage collection process. A *version control mechanism* allows the user to recover previous versions of an object.

To account for the validity of library objects, the knowledge base supports *dependency tracking*, which will enable a user to check if theorems are valid wrt. a specific set of rules, axioms, and proof procedures.

In principle, the library does not impose any predefined structure. All visible structure, e.g. the directory structure as observed by the NUPRL 5 navigator (See Chapter 4), is generated by structure objects that are explicitly present in the library and can be customized by the user.

To prevent name clashes, the library distinguishes between objects and the names that users choose to denote them. The latter are just display versions of internal names. The can be changed without affecting the object itself.

User Interfaces

The main user interface of NUPRL 5 is the *navigator*. It communicates with the knowledge base by sending and receiving abstract terms. While displaying and editing these terms it presents them as directories, theorems, definitions, proofs, or mathematical expressions – depending on structural information found in the library. For the user, it provides the functionality of a *structure editor*: the user can mark subterms and edit slots in the displayed term and then cause the navigator to send the result back to the library, which processes the result while the user may continue to work with the editor. Again, structural information in the library determines whether the abstract terms received by the knowledge base are interpreted as a commands to store or retrieve data, as tactics calling a refiner, or as utility functions.

In addition to the navigator, NUPRL 5 provides emulations of the editors used in the NUPRL 4 system, as well as valuable extensions for facilitating proof browsing, merging, replaying and accounting. There is also a web front end [Nau98] that allows external users to browse the NUPRL library remotely.

Inference Engines

The NUPRL 5 inference engine refines proof goals by executing ML code that may include references to library objects, particularly to the inference rules and tactics stored in the knowledge base. It applies the code to a given proof goal that it receives as an abstract term and returns the resulting list of subgoals back to the library. Based on the validations given in the rule objects it can also extract programs from proofs and evaluate them. The inference mechanism is fairly straightforward and compatible with the one in NUPRL 4.

As an alternative one may invoke the MetaPRL refiner [Met], a modularized version of NUPRL's inference engine implemented in OCaml, which is significantly faster due to improvements in rewriting and evaluation. The communication between NUPRL 5 and MetaPRL utilizes the MathBus design [Mat].

We are in the process of connecting a variety of external refiners such as a constructive first-order theorem prover [KOSP00], the HOL system (via Maude [CDE+99b]), Mathematica, and Isabelle [Nau99]. We will also emulate the refiner of NUPRL 3 in order to be able to restore older theories that did not survive the transition from NUPRL 3 to NUPRL 4.

1.2 Purpose of this Manual

This manual is a reference manual for version 5 of the NUPRL system. It is aimed at beginning and intermediate users of the system. NUPRL 5 is written mostly in Common Lisp, but uses some extensions that require Lucid or Allegro Lisp. It runs on Unix-based workstations that use the X window system.

Note, that this manual is still under development and incomplete. Additional online documentation can be found on the Web at the URL `http://www.nuprl.org/html/nuprldocs.html`, in the directory `/home/nuprl/nuprl5/doc/` of the installed system, and in objects such as `doc: ref editor` and `doc: navigator use`, while the system is running.

The original NUPRL book *Implementing Mathematics with the Nuprl Proof Development System* [CAB+86], also available on the Web at the URL `http://www.nuprl.org/book/doc.html` is still a good background reference. However, one has to keep in mind that the system itself has been changed and extended substantially since the book was published. None of the tutorials given in the book will work in NUPRL 5. The "reference" portion of the book is superseded by this reference manual, but contains some useful examples and discussions of tactic writing that are not reproduced here. The "advanced" portion of the book deals with application methodology, gives some extended examples of mathematics formalized in Nuprl, and also describes some extensions to the type theory which have not been implemented.

1.3 Tips for Beginning NUPRL 5 Users

We recommend that you run through the brief tutorial in chapter 2 before trying to do anything else with the system.

In learning to use the navigator as well as the proof and term editors, check out all the mouse commands and the buttons that are provided. Many editing operations can be done most easily with the mouse and the buttons. Familiarize yourself with the standard NUPRL theories that are already present in the library. Existing theories are an excellent resource for learning about how to structure theories and how to write proofs.

The NUPRL ML manual in Appendix B contains a tutorial in the use of ML (Appendix B.2). Use this as an introduction to ML. Daring users may also take a look at the `.ml`-files in the directory `/home/nuprl/nuprl5/lib/ml/standard`, which contain the implementation of the existing tactics collection and can be used as examples for writing tactics.

We recommend that fairly early on, you at least browse through this manual, familiarizing yourself with the general contents of each chapter. This will help you know where to look if you have questions.

1.4 Conventions

We give the conventions we use in this manual for presenting user input and NUPRL output. Input which you should type is presented typewriter font. For example `this is in typewriter font`. The following symbols are also used:

- $\boxed{\text{SPC}}$ for the space-bar.
- ↵ for the Return key (sometimes marked as Enter).
- $\boxed{\text{LFD}}$ for the linefeed key.
- ⇆ for the Tab key.
- $\boxed{\text{DEL}}$ for the delete key (sometimes marked as Rubout). On some keyboards the $\boxed{\text{BACKSPACE}}$ has the same effect.
- $\boxed{\text{LEFT}}$, $\boxed{\text{MIDDLE}}$, and $\boxed{\text{RIGHT}}$ for the left, middle, and right mouse button.

Modified keys are presented as follows:

- $\langle$C-$x\rangle$ read as "control x". Hold down a control key and simultaneously press key x.
- $\langle$M-$x\rangle$ read as "meta x". Hold down a meta key and simultaneously press key x.
- $\langle$C-M-$x\rangle$ read as "control meta x". Hold down both a control key and a meta key, and simultaneously press key x.
- $\langle$S-$x\rangle$ read as "shift x". Hold down a shift key and simultaneously press key x.

Note that x can be either a keyboard key *or* a mouse button; for example both $\langle$C-`a`$\rangle$ and $\langle$M-$\boxed{\text{RIGHT}}\rangle$ are valid modified keys. On some keyboard's (for example, those of Sparc-stations) the usual meta keys are the keys marked $\diamond$ either side of the space-bar, while on PC keyboards this key is often marked as Alt. The $\langle$S-$x\rangle$ modifier is only used with non-printing characters (for example, ↵).

When we say "click $\boxed{\text{LEFT}}$" on some part of a window, we mean that the mouse cursor should be pointed at that part, and then the $\boxed{\text{LEFT}}$ button should be pressed.

Be aware that occasionally Nuprl can be quite slow to respond to keystrokes, sometimes taking several seconds. Don't hold keys down till you get a response. You might easily make the keys autorepeat, which could be rather annoying.

For clarity when presenting input which a user might type, or output which Nuprl generates, we sometimes enclose the text in special ⌞⌟ quotes. For example ⌞`this is example output`⌟.

1.5 Structure of this Manual

Chapter 2 gives a tutorial-like overview of the essential features of the Nuprl 5 system. Novice users should run through this tutorial before trying to do anything else.

Chapter 3 describes how to install, start, and exit the system. It also give a few hints on customization and basic troubleshooting.

Nuprl's windows are at the "top-level" in the X environment and come with their own context-specific editors. The two main windows – the *navigator* and the ML *top loop* – are described in Chapter 4 while Chapters 5 and 6 describe the editors for *term* and for *proof windows*.

The following chapters describe Nuprl's support for the formalization of mathematical concepts and proofs. In chapter 7 we explain how to extend the logical language of Nuprl by abstract definitions and how to modify the visual presentation of formal material through display forms. Chapter 8 describes the structure of basic inferences in Nuprl as well as the most important *tactics* for automated reasoning.

The appendices describe important background information such as the type theory of Nuprl (Appendix A) and Nuprl's meta-language ML (Appendix B).