The Nuprl Open Logical Environment

S. F. Allen, R. L. Constable, R. Eaton, C. Kreitz, and L. Lorigo *

Department of Computer Science, Cornell-University, Ithaca, NY 14853-7501
{sfa,rc,eaton,kreitz,lolorigo}@cs.cornell.edu

Abstract. The Nuprl system is a framework for reasoning about mathematics and programming. Over the years its design has been substantially improved to meet the demands of large-scale applications. Nuprl LPE, the newest release, features an open, distributed architecture centered around a flexible knowledge base and supports the cooperation of independent formal tools. This paper gives a brief overview of the system and the objectives that are addressed by its new architecture.

1 Introduction

The Nuprl proof development system [C+86] is a framework for the development of formalized mathematical knowledge as well as for the synthesis, verification, and optimization of software. The original system was 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 it has been used in increasingly large applications in mathematics and programming, such as verifications of a logic synthesis tool [AL93] and of the SCI cache coherency protocol [How96] as well as the verification and optimization of group communication systems [KHH98,Kre99,L+99].

Over the years it has turned out that the rapidly growing demands for formal knowledge and tools cannot be met by a single closed system anymore. Automatic tools such as decision procedures, fully automatic theorem provers, proof planners, rewrite engines, model checkers, and computer algebra systems have been very successful in their respective areas, but have limited application domains. Proof assistants like Nuprl, Isabelle [Pau90], HOL [GM93], PVS [O+96], and Ωmega [B+97] are more general but at a lesser degree of automation. Each of these systems has accumulated a substantial amount of formalized knowledge in its respective formalism, but no system contains all the currently available formal knowledge. A variety of user interfaces have been developed for these systems each with its own strengths and weaknesses.

* Part of this work was supported by DARPA grant F 30620-98-2-0198

D. McAllester, ed. *17th International Conference on Automated Deduction*, LNAI 1831, pp. 170–176, ©Springer Verlag, 2000.

These observations led to an entirely new design of the Nuprl system that shall replace the monolithic architecture of current theorem proving environments. The Nuprl LPE (logical programming environment) is an *open*, distributed architecture that integrates all its key subsystems as *independent* components and, by using a flexible knowledge base as its central component, supports the interoperability of current proof technology.

In the following we shall briefly discuss the key issues that shall be addressed by Nuprl LPE and describe its architecture as well as the available components.

2 Design Objectives

Besides preserving and expanding the strengths of the existing Nuprl system, the new design of the Nuprl LPE is based on the following objectives.

Interoperability: The Nuprl LPE shall provide a platform for the cooperation of proof systems and a common knowledge base that makes formal theories available to the individual systems. Special support for computational logics shall be offered, but other logics shall be accommodated as well.

Optimization and Productivity: To optimize software reuse and system maintenance, the key components of the Nuprl LPE have to be independently operating programs that communicate using a protocol. This will increase the system's productivity, as several inference engines can be run in parallel or even off-line while the user continues to work on other proof goals.

Accountability: As the Nuprl LPE shall accomodate a variety of logics, there cannot be an absolute notion of correctness anymore. Instead, the extent to which one may rely upon formalized knowledge in the library must be accounted for. Justifications for the validity of proofs depend upon what rules and axioms are admitted and on the reliability of the inference engines employed. The design has to make sure that such information can be easily exposed to determine which proofs are valid in a particular logic.

Information Preservation: The system has to make sure that knowledge cannot be destroyed or corrupted if a user erroneously overwrites a proof or if the system crashes before a proof could be saved. The system must guarantee that such information can always be recovered.

Large Scale Object Management: The system should use abstract object references rather than traditional naming schemes. This is invaluable for merging mass libraries, where name collisions are inevitable, and also for performing context-specific tasks.

3 The Nuprl LPE Architecture

Figure 1 illustrates the distributed open architecture of Nuprl LPE. The system is organized as a collection of communicating processes that are centered around a common knowledge base, called the *library*. The library contains definitions, theorems, inference rules, meta-level code (e.g. tactics), and structure objects

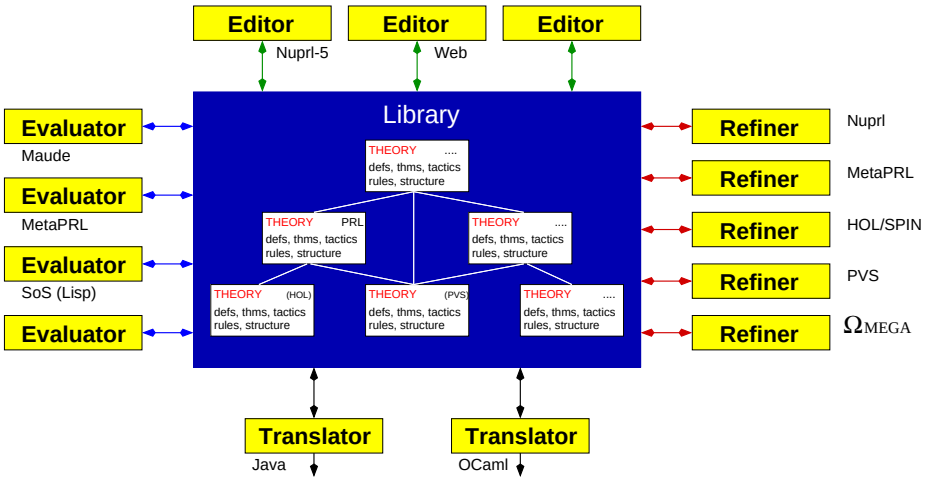

Fig. 1. Nuprl LPE distributed open architecture

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, major systems like HOL, PVS, Ωmega, or SPECWARE [SJ95], decision procedures, first-order provers, Mathematica [Wol88], and the Maude [C$^+$99b] rewrite engine, and to have them cooperate through the library, which stores the formal knowledge required by these tools. 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 restart the whole system, as one would have to do in monolithic architectures.

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

The Nuprl LPE provides special support for computational and constructive logics but it can accommodate other logics equally well. Its open architecture makes it possible that different systems with different formalisms and representation structures cooperate through a common knowledge base that can store all these informations. It is obvious that translations between different formalisms need to be developed to make such a cooperation possible and that several theoretical issues need to be addressed for each of them (see e.g. [How96,FH97]).

But the Nuprl LPE provides the necessary infrastructure for these translations. They only have to operate on the formal knowledge stored in the library and can be provided as independent external processes that are invoked as necessary. Translations can also be used in a transitive fashion, e.g implementing the translation between Maude and Nuprl [C$^+$99a] automatically gives us access to all formalisms that have been translated into Maude.

In the current standard configuration, which we call Nuprl 5, 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 library contains all the definitions, theorems, inference rules, and tactics of Nuprl 4 as well as the structure objects that emulate the Nuprl 4 system architecture. The Nuprl 5 editor is capable of interpreting these structure objects while displaying and editing proofs and terms. The Nuprl 5 refiner is able to interpret inference rules and the ML code of the tactics. In the following we will describe the individual components more specifically.

The Knowledge Base. The knowledge base is based on a transaction model for entering and modifying objects. Changes to objects, e.g. the effects of editor commands or inference steps, are immediately committed to the library. This makes sure that knowledge doesn't get lost in case of a system failure, which could happen in systems that keep newly developed knowledge in memory until it is explicitly saved to disk. 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. This protects user data from being corrupted or destroyed erroneously and enables a user to create several proofs of the same theorem.

To account for the validity of library objects, the knowledge base supports *dependency tracking*. For this purpose a variety of information is stored together with an object, e.g. the logical rules and theorems on which it depends, the exact version of the refiners that were used to prove it, timestamps, etc. This information will help a user to validate theorems that rely on knowledge created by several systems, provided that the conditions for *hybrid validity* wrt. the underlying logics are well understood and stored in the library. For instance, a theorem referring to lemmata from Nuprl (constructive type theory) and HOL (classical higher order logic) would be marked as constructively valid if the HOL theorems involve only decidable predicates.

Apart from abstract links from terms to library objects the library does not impose any predefined structure. All visible structure, e.g. the directory structure as observed by the Nuprl 5 editor, is generated by structure objects that are explicitly present in the library. This allows exploiting the structure of the library and modifying it without having to change the representation of already stored knowledge. Structure, as we understand it, is only a matter of external presentation, not of internal representation.

For the same reason, there is a separation between objects and the names that users choose to denote them. Technically, the visible name is just a display version of the internal name, which makes it possible to chose the same name for different objects without creating internal name clashes and to disambiguate the display as needed.

The absence of a predefined library structure is a prerequisite for integrating formal knowledge from other systems besides Nuprl without requiring these systems to change their representation structure. The "only" thing that needs to be done is to emulate this structure in Nuprl's knowledge base.

User Interfaces. The main user interface of Nuprl 5 is the Nuprl 5 *navigator*. Its communication with the knowledge base is based on sending and receiving abstract terms. While displaying and editing terms it interprets the corresponding structure objects and displays them as directories, theorems, definitions, proofs, or mathematical expressions, sometimes opening new windows for this purpose. 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.

The meaning of the abstract terms received by the knowledge base is determined by the structure objects already present. It could be interpreted as a command to store the term, to execute a tactic which subsequently calls one or several refiners, to open a proof editor window, or to send another term to be displayed. Obviously, this process involves a lot of management information in the terms being sent that is usually not shown to the user. However, the user has the right to edit (almost) all structure objects as well and thus customize the appearance of the information presented by the editor.

The Nuprl 5 editor is capable of interpreting objects as commands. A very convenient feature resulting from that is a hyperlink mechanism where clicking on that term causes the corresponding object to be raised, but more general ML expressions can be executed as well. This enables the user to trace back definitions of logical expressions and tactics or to customize the editor by adding buttons for common commands.

In addition to the Nuprl 5 navigator, Nuprl 5 provides emulations of the editors used in the previous release of Nuprl in order to ensure upward compatibility, 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 without having to install the whole system.

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. In the process it may invoke decision procedures and proof checkers. 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, that can run up to 100 times faster due to improvements in rewriting and evaluation. The communication between Nuprl LPE and MetaPRL utililizes the MathBus design [Mat].

We are currently working on connecting a variety of external refiners such as a constructive first-order theorem prover [K$^+$00], the HOL system (via Maude [C$^+$99a]), Mathematica, and Isabelle [Nau99]. We will also emulate the refiner of Nuprl 3 in order to be able to restore older theories that had not been migrated during the transition to Nuprl 4.

4 Progress and Availability

Nuprl LPE is the result of more than 25 years of experience with mathematical proof assistants. We have completed the implementation of the basic Nuprl LPE system, which provides a platform for the cooperation of a variety of proof systems through its open distributed architecture.

Using the Nuprl LPE infrastructure we have implemented the Nuprl 5 system consisting of the Nuprl 5 editor, refiner, evaluator, and the Nuprl 5 core library. The latter contains the terms and rules of the Nuprl type theory as well as the standard theories and tactics, which were migrated from Nuprl 4 to Nuprl 5. Besides the Nuprl 5 refiner a user may also invoke the MetaPRL refiner.

In addition to that we have migrated most of the user-defined theories from Nuprl 4 to Nuprl 5 and are currently testing the behavior of the new system under large scale applications like the verification of communication systems. Experience shows that the separation of library, editor, and refiner makes Nuprl 5 more efficient than its predecessor Nuprl 4, because it can take advantage of multiprocessor machines or a network of computers and run several competing refiners to solve a goal. We also observed an increased productivity of the system's users, who can now work on other tasks while a refiner solves a complex goal.

The completion of the basic Nuprl LPE system enables us to increase the system's capabilities by adding new editors, refiners, evaluators, or translators to the system. The open distributed architecture opens the door for a variety of research topics which can be turned into practically useful components as soon as their theoretical background has been explored.

Nuprl LPE is written mostly in Common Lisp, but uses some extensions that require Lucid or Allegro Lisp. An executable copy running under Linux is available at `http://www.cs.cornell.edu/Info/Projects/NuPrl/nuprl5/index.html`.

References

[AL93] M. Aagaard & M. Leeser. Verifying a logic synthesis tool in Nuprl. *CAV-93*, LNCS 663, pp. 72–83. Springer, 1993.

[B$^+$97] C. Benzmüller et. al. Ωmega: Towards a mathematical assistant. *CADE-14*, LNAI 1249, pp. 252–256. Springer, 1997.

[C+86] R. Constable et. al. *Implementing Mathematics with the NuPRL proof development system*. Prentice Hall, 1986.

[C+99a] M. Clavel, F. Duran, S. Eker, J. Meseguer, M.-O. Stehr Maude as a formal meta-tool. *FM'99*, LNCS 1709, pp. 1684–1703, Springer, 1999.

[C+99b] M. Clavel, F. Duran, S. Eker, P. Lincoln, N. Marti-Oliet, J. Meseguer, J.F. Quesada. The Maude system. *RTA'99*, LNCS 1631, pp. 240–243, Springer, 1999.

[FH97] Amy Felty and Douglas Howe. Hybrid Interactive Theorem Proving using Nuprl and HOL. *CADE-14*, LNAI 1249, pp. 351–365. Springer, 1997.

[GM93] M. Gordon & T. Melham. *Introduction to HOL: a theorem proving environment for higher-order logic*. Cambridge University Press, 1993.

[How96] D. Howe. Importing mathematics from HOL into NuPRL. *Theorem Proving in Higher Order Logics*, LNCS 1125, pp. 267–282. Springer, 1996.

[Jac94] P. Jackson. *The Nuprl Proof Development System, Version 4.2*. Cornell University. Department of Computer Science, 1994.

[KHH98] C. Kreitz, M. Hayden, J. Hickey. A proof environment for the development of group communication systems. *CADE-15*, LNAI 1421, pp. 317–332. Springer, 1998.

[Kre97] C. Kreitz. Formal reasoning about communication systems I: Embedding ML into type theory. Technical Report TR97-1637, Cornell University. Department of Computer Science, 1997.

[Kre99] C. Kreitz. Automated fast-track reconfiguration of group communication systems. *TACAS-99*, LNCS 1579, pp. 104–118. Springer, 1999.

[K+00] C. Kreitz, J. Otten, S. Schmitt, B. Pientka. Matrix-based Constructive Theorem Proving. *Intellectics and Computational Logic*. Kluwer, 2000.

[L+99] X. Liu, C. Kreitz, R. van Renesse, J Hickey, M Hayden, K. Birman, R. Constable. Building reliable, high-performance communication systems from components. *SOSP'99*, Operating Systems Review 34(5):80-92, 1999.

[Mat] The MathBus Term Structure.
`http://www.cs.cornell.edu/simlab/papers/mathbus/mathTerm.htm`.

[Met] Metaprl home page.
`http://ensemble01.cs.cornell.edu:12000/cvsweb/meta-prl`.

[ML84] P. Martin-Löf. *Intuitionistic Type Theory*, Bibliopolis, 1984.

[Nau98] P. Naumov. Publishing formal mathematics on the web. Technical Report TR98-1689, Cornell University. Department of Computer Science, 1998.

[Nau99] P. Naumov. Importing Isabelle formal mathematics into Nuprl. Technical Report TR99-1734, Cornell University. Department of Computer Science, 1999.

[O+96] S. Owre et. al. PVS: Combining specification, proof checking and model checking. *CAV-96*, LNCS 1102, pp. 411–414, Springer, 1996.

[Pau90] Lawrence C. Paulson. Isabelle: The next 700 theorem provers. *Logic and Computer Science*, pp. 361–386. Academic Press, 1990.

[SJ95] Y. V. Srinivas and Richard Jüllig. SPECWARE: Formal Support for composing software. In *Mathematics of Program Construction*, 1995.

[Wol88] S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer*. Addison Wesley, 1988.