# System Description: A Nuprl-PVS Connection: Integrating Libraries of Formal Mathematics*

Stuart Allen, Mark Bickford, Robert Constable, Richard Eaton, Christoph Kreitz

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

**Abstract.** We describe a link between the Nuprl and PVS proof systems that enables users to access PVS from the Nuprl theorem proving environment, to import PVS theories into the Nuprl library, and to browse both Nuprl and PVS theories in a unified formal framework. The combined system is a first step towards a digital library of formalized mathematics that can be shared and used in complex applications.

## 1 Introduction

Over the past 20 years large collections of formal mathematical knowledge have been created by users of proof systems such as Coq, HOL, Isabelle, Ωmega, Nuprl, and PVS. These collections include a substantial amount of general mathematical results that support a variety of applications, particularly in the design and verification of reliable hardware and software. However, since the formal material is represented in a variety of formalisms and data structures, it has not yet been possible to present it in a common repository, which in turn prevents formalized mathematics from being shared and more widely usable.

The formal digital library (FDL) now supporting the Nuprl system [1,2,3] provides the infrastructure for building such a common repository. It includes a uniform term language for representing the formalisms of proof systems. It incorporates mechanisms for storing, certifying, browsing, organizing, searching, and annotating formal material. It can be connected to an arbitrary number of inference engines, user interfaces, and proof systems, which may contribute new formal knowledge to the repository and incorporate it in other digital environments.

In the standard Nuprl 5 interface the FDL contains the definitions, theorems, inference rules, and tactics of Nuprl's constructive type theory as well as all the application theories developed in it. One or several Nuprl refiners serve as inference engines while users may interact with the system through the highly visual Nuprl editor. The library has also been connected to two external refiners: the MetaPRL proof engine [4], a modularized version of Nuprl's refiner based on fast rewriting, and JProver [7], a complete automatic theorem prover for intuitionistic and classical first-order logic.

The link between Nuprl and PVS [5], which we will describe here, connects two major proof systems that are based on different formal theories: constructive and classical type theory. It makes it possible to access the proof engines and knowledge bases of Nuprl and PVS through a uniform formal framework and to use the joint system in a variety of new applications.

* This work was supported by ONR Grant N00014-01-1-0765 (Building Interactive Digital Libraries of Formal Algorithmic Knowledge) and by NSF Grant CCR 0204193 (Proof Automation in Constructive Type Theory).

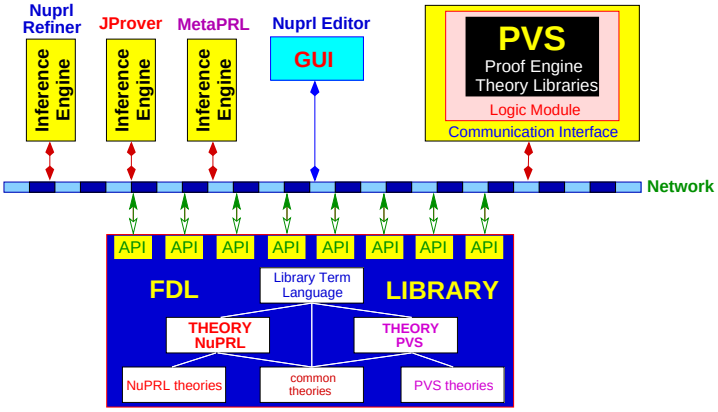


**Fig. 1.** Architecture of Nuprl in connection with PVS

## 2 Communicating between PVS and Nuprl

PVS [5,6] is a proof system based on a classical, typed higher-order logic. It supports interactive reasoning and proof scripts that build sequent-style inferences from primitive inference rules, induction, rewriting, and decision procedures. In addition to its proof engine it contains libraries of formal mathematical theories that have been build by users of the system.

Figure 1 depicts the architecture of the link between PVS and a standard configuration of Nuprl. PVS is treated as a black box system and is wrapped by two Lisp modules: a *communication interface* for communicating terms between PVS and the library and a *logic module* for translating these the logical language of PVS into these terms.

Since all data and commands of the FDL are expressed as abstract terms, Nuprl communicates with PVS through terms of the library's uniform term language (see [3, Chapter 5]). The communication interface implements a protocol for sending these terms over an INET socket to an application interface of the library and for receiving terms through that channel.

The logic module is a Lisp package that provides an implementation of the library's uniform term language as PVS data structure, a representation of the logical language of PVS as abstract library terms in that implementation, and an algorithm that converts PVS output such as formulas, sequents, declarations, proof scripts, theorems, etc. into their representation as abstract library terms.[1]

The combination of these two modules makes it possible to have a Nuprl user issue commands to the PVS proof system, to send genuine PVS expressions from PVS to Nuprl and to store them as objects of the formal digital library without further modifications.

For users of the joint system we also have provided a collection of display forms (see [3, Chapter 7.2]) that make sure that library representations

[1] Note that the algorithm does not convert between logics but only between the data structures of the respective systems.

of PVS expressions are displayed like genuine PVS terms. The display form `number-pvs_df`, for instance, makes the abstract term `number{pvs:t,4:n}()` appear on the screen as the number 4. These display forms represent the core theory of PVS in the library.

## 3 Using PVS from Nuprl

### 3.1 Invoking the PVS Proof Engine

While in principle it is possible to invoke the PVS proof engine on any node of an arbitrary proof tree, using PVS as a black box makes sense only for top-level goals that represent proper PVS statements. Currently, we call the PVS proof engine only on theorems in existing PVS theories that can be proven without further interaction. Issuing the command `pvs_proof_of_lemma` *`theory lemma`* to the Nuprl library will result in the following activities.

- Nuprl sends the names of the theory and the lemma as well as the command to prove the lemma to the wrapped PVS system.
- Using hooks into the basic PVS functions the wrapper modules cause PVS to retrieve the lemma from the corresponding file and to generate a full proof for it, possibly executing an already existing proof script.
- Once the PVS proof is complete, the generated proof tree can be found in the PVS variable `proof_state`. The logic module converts the contents of this variable into an abstract term and sends this term to the Nuprl library.
- Upon receiving the abstract proof term, the library constructs a proof tree, storing references to the executed PVS steps as justification for each inference. Users may now browse the generated proof with the Nuprl proof editor.

It should be noted that the library has to consider PVS a *trusted refiner*, which means that it cannot check the inferences by other means than referring to PVS. This is in contrast to the Nuprl-JProver connection [7], where the proofs generated by JProver may be converted into genuine Nuprl proofs.

To invoke the PVS proof engine incrementally at a given proof node one would have to send both the proof goal and the inference steps to PVS. However, since PVS usually interacts with files and expects theorems to belong to some theory, one would also have to turn the goal into a proper PVS statement, turn the inference steps into a PVS proof script for this statement, wrap the statement with appropriate theory declarations, and then invoke PVS on the new theory. We will investigate this approach in the future.

### 3.2 Migrating Formal Theories

PVS users have developed significant amounts of formal knowledge about the formal specification and verification of software. However, PVS does not store all the inference steps of successful proofs but only the corresponding proof script. Migrating a theory from PVS to the formal digital library therefore requires us to reconstruct the missing information by replaying all the proofs of that theory in PVS and rebuilding the corresponding proof tree in the library.

Migrating theories, however, means more than simply replaying proofs from scratch. In a theory, proofs often contain references to definitions, axioms, and lemmata of the theory. As the latter are represented by separate library objects, these references have to be converted into links to the appropriate objects in order to justify the corresponding proof steps. Importing a PVS theory into the FDL thus involves the following activities.

- First all the declarations of the PVS theory are sent to formal digital library, which represents them as objects of a fresh library theory.
- To make these objects accessible to other migrated objects the library also constructs a object containing links between these objects and a string representing their name in PVS.[2]
- For each PVS theorem the library builds a proof object with the theorem's statement as root, links this object to a string representing its PVS name, and then rebuilds its proof by replaying the stored proof script as described in Section 3.1.
- If a proof cannot be rebuilt, because it refers to lemmata that have not yet been constructed, it is marked as failed and replayed at a later time.

All these steps are performed by the function `PVS_import` *`PVS-theory-name`*. We have used this function to migrate all 79 theories of the PVS prelude, the complete libraries on bitvectors, finite sets, arrays, number and graph theory, real analysis, etc and are in the process of importing the remaining theories of NASA Langley PVS library. Migrating PVS content to the formal digital library makes the library a common repository for both Nuprl and PVS users.

### 3.3 Browsing PVS theories in Nuprl

To browse PVS theories in the formal digital library, users may take advantage of the highly visual Nuprl editor, which enables them to follow hyperlinks between objects and components of formulas that were established during the migration process. This has certain advantages for novice users that are not yet familiar with the emacs commands needed to control the original PVS system. On the other hand, the display forms stored in the library make sure that theories and theorems are displayed (almost) the same way as in PVS.

To start browsing PVS theories, move the navigator [3, Chapter 4] into the directory `[theories; pvs; theories]`.

- To open a PVS library (e.g. `number_theory`), middle-click on its name.
- To open a specific PVS theory (e.g. `primes`), middle-click on its name. This pops up a new window showing the theory term, which appears in the same way PVS shows the corresponding theory in emacs.
- To view a the proof script of a lemma (e.g. `prime-factors`), right click on name of the lemma in the theory. The proof script is displayed in a new window and displayed as a list of ML tactics connected by tacticals (PVS would show them as nested list of Lisp commands).

[2] The reason for this level of indirection is that library objects have *abstract* identifiers. See [2, Chapter 3.1] for a detailed discussion

– Clicking on the line `(*to see full proof, right click here*)` will show the complete proof tree, including all the subgoals created by each inference step. Users can walk through this proof using arrow keys as described in [3, Chapter 6]. Full proofs cannot be viewed in the PVS system.
– Within a theory, users may also click on declarations like the type of declared variables or imported theories. This will pop up windows displaying the corresponding definitions or theories.

## 4 Progress and Availability

The Nuprl-PVS link is a first step towards a common repository of formalized mathematics that can be shared among automated proof systems. The current version is still at an experimental stage and requires expertise about Nuprl and PVS. A demonstration of the joint system will be given at the conference.

PVS can be downloaded from the PVS home page `http://pvs.csl.sri.com`. The Nuprl system, including the library containing all the imported PVS theories, can be found at the Nuprl home page `http://www.nuprl.org`. The wrappers for PVS and instruction for installing the link between the two systems will be posted in the near future.

Currently, the FDL keeps the theories of Nuprl and PVS separate with no interaction between them. However, as classical and constructive type theory overlap on decidable expressions, it is desirable to identify subsets of Nuprl and PVS theories that have the same semantics and to link them within the library. This would allow using the PVS proof engine on certain Nuprl proof goals and vice versa and thus strengthen the automated reasoning capabilities of the joint system. We will investigate such semantical links in the future.

We also plan to migrate formal content developed with proof systems such as HOL, Coq, Isabelle, and possibly Larch to the formal digital library and to develop mechanisms for making the collected formal material available on the web.

## References


1. S. Allen, R. Constable, R. Eaton, C. Kreitz, L. Lorigo. The Nuprl open logical environment. *17th Conference on Automated Deduction*, LNAI 1831, pages 170–176. Springer Verlag, 2000.
2. S. Allen, M. Bickford, R. Constable, R. Eaton, C. Kreitz, L. Lorigo. FDL: A prototype formal digital library. Cornell University. Department of Computer Science, 2002.
3. C. Kreitz. *The Nuprl Proof Development System, Version 5: Reference Manual and User's Guide*. Cornell University. Department of Computer Science, December 2002.
4. Metaprl home page. `http://metaprl.org`.
5. S. Owre, S. Rajan, J. M. Rushby, N. Shankar, M. K. Srivas. PVS: Combining specification, proof checking and model checking. *Computer-Aided Verification*, LNCS 1102, pages 411–414. Springer Verlag, 1996.
6. PVS home page. `http://pvs.csl.sri.com`.
7. S. Schmitt, L. Lorigo, C. Kreitz, A. Nogin. JProver: Integrating connection-based theorem proving into interactive proof assistants. *International Joint Conference on Automated Reasoning*, LNAI 2083, pages 421–426. Springer Verlag, 2001.