Logical Aspects of Digital Mathematics Libraries (extended abstract)

Stuart Allen[1⋆], James Caldwell [∗2], and Robert Constable[1∗]

[1] Department of Computer Science, Cornell University, Ithaca NY 14853
[2] Department of Computer Science, University of Wyoming, Laramie WY 82071

1 Introduction

The organization of mathematical knowledge on a large scale is an idea whose time has come. In a recent paper in the Bulletin of Symbolic Logic, prospects for mathematical logic in the next century were outlined. Among three predictions[1] made there for proof theory and logic in computer science in the next century is the following:

> *Computer databases of mathematical knowledge will contain, organize, and retrieve most of the known mathematical literature by* 2030 ± 10 *years.*[6, pp.181]

Steps toward the systematic formalization of mathematics goes back more than a hundred years to Frege, Russell, Hilbert and others, but it is only the advent of a diverse collection of computer based technologies that makes the possibility of realizing this goal truly possible. For the past thirty years and more, the computer science community has been constructing the theoretical and technical infrastructure to support this effort, only now are enough of the pieces in place to allow a credible attack on the problem. As Constable said:

> [we have now created a digital computer based proof technology that will] *enable a diverse community of mathematicians, computer scientists and educators to build a new artifact — a globally distributed library of formalized mathematics.*[9]

The solution will include (at least) the best of data-base techniques, distributed systems and web technology, and of course will build on the significant developments in theorem proving and symbolic computation.

For more than twenty years we have been designing, implementing and applying constructive systems to the formalization of algorithmic mathematics and to modeling and verifying complex computer systems. Our production system, Nuprl [7] has been used in several major applications [15, 5, 14, 1] and has been used to solve difficult problems in mathematics [17, 16, 13]. The mathematics formalized in the system have been used to teach computer science and logic courses [8]. Many of the questions that arise in an effort to build a global digital library of mathematics arise also in the formal setting. We see interesting possibilities for sharing ideas and methods with the larger effort.

The experimental results that support this article are based on the formalization of particular fragments of computational mathematics, but the results are general. In this paper we focus on capabilities of digital mathematics libraries that are enabled by formalism. Specifically we report results on these topics.

First, our formalism is distinctive in that formal proofs are readable. In particular they can be read at various levels because of their structure, and they support the other formal structures called readings or glosses that present aspects of the proof. These structures allow us to collect sets of inference steps that serve various uses.

Second, formal proofs can be manipulated. They can be mapped to HTML and projected automatically to the Web. They can be reorganized. They can be linked together into readings of theories.

Third, the formalism allows semantic processing of proofs. In particular we can track dependencies on definitions, lemmas and proof tactics. A formal semantics serves as a basis for relating the formal proof

⋆ Part of this work was supported by ONR N00014-01-1-0765

[1] The other two predictions were related to the solution of the $P \stackrel{?}{=} NP$ problem and prospects for "limited but significant success in artificial intelligence.".

to informal accounts of the ideas, and we can use the semantics to transition from a formal setting to the corresponding informal one.

Fourth, the formalism allows us to state precise mathematical properties of proofs and theories. This is the basis for checking their correctness, but also for investigating questions such as how many results depend on a certain rule or a certain axiom.

There are interesting theoretical consequences of our way of providing library services based on the formalism, but in this paper we focus on reporting the mechanisms and the lessons learned from using them. The paper will refer to the on-line library and to features of it that readers of this paper can examine.

Our latest implementation is an open Logical Programming Environment (LPE) [3]. One of the key components of our LPE is a *database of thousands of formally proved mathematical statements* of general mathematical interest, many related to data types and algorithms. Our logic expresses computational concepts, and it keeps track of computational content in assertions and proofs. Moreover, the proofs are readable and accessible to a wide audience via the Web [18].

2 General Context Of Our Work

As we have thought about how our Library might be expanded to become a *global interactive information resource*, we have identified key technical challenges and specific intermediate objectives. Specifically we are developing approaches to the problem of accounting for correctness and truth in a library that allows multiple logics and multiple theorem provers, for knowing exactly what a result depends on, for combining sub-libraries and for performing a variety of routine operations on libraries such as searching and browsing. We are also developing very advanced operations on theories such as soundly translating among them, generalizing, specializing and reflecting them. These are operations on theories stored as objects in the Library and operations on code in these theories. We plan to use the computational contents of proofs as components of programs in other programming languages in a consistent way and to provide interaction with the Library using the Web. These goals generate many interesting technical problems, several of which we discuss below .

Some basic facts about formal digital libraries are clear. The contents should include assertions written in a formal logical language, perhaps in several of them. Libraries should contain fully readable proofs, and memory is no longer a barrier to storing complete formal proofs. Readable proofs are valuable content items because they provide explanation. We can also automatically manipulate existing proofs to obtain new ones, translate between them, transform theories to create new ones, and we are increasing our capability for generating natural language versions of proofs from the formal proofs [12].

In order to create complete formal proofs, we need theorem provers that exploit a mix of interactive and fully automatic methods. The automatic ones comprise decision procedures, model checkers, and heuristic provers. Our systems combine these; recently we added the JProver [21], a fast prover for Intuitionistic first-order logic. Nuprl is connected to the JProver through MetaPRL. The interactive components call upon these automatic methods and also execute high level descriptions of inference steps in the form of tactics, derived rules and proof plans.

Among the less obvious insights into the nature of formal libraries of mathematics is the realization that it is important to store theorem proving *methods* in the library. Another less obvious insight is that the library is much more useful if it can express formal relationships among theories, which in turn requires *reflection* and meta-reasoning. A third subtle insight is that there are major benefits to organizing the base level of the library as a relatively unstructured collection of theorems and claims connected by logical dependency. On this base we can overlay many independent structures or *readings* of the library. These readings can be like books or theories or lectures or system documentation or structures yet imagined, enabled by making the formal content accessible to "intelligent" computer systems. Readings are themselves stored in the library.

Our perspective is based on extensive experience working in our system and we summarize a number of basic principles arrived at through these efforts.

Interoperability: Semantic mappings between formal systems is far preferable to a (we believe, doomed) attempt to find a single unified formal system. No one formal system is adequate to the task, this is accounted for as much, if not more, by philosophical, social, and political reasons as for technical ones.

Even within the context of the Nuprl system, we have interacted with a number of different theories [see Fig.1].

Interaction: The objects used to represent the mathematics must allow interactive inspection. These objects include not only axioms, proof rules, proofs, definitions, theorems, and theories but must also accommodate inspection of algorithmic content which includes inference engines, tactics, and decision procedures. A key form of interation is provided by relations between formal and informal presentations.

Accountability: A user has the right to know what each result depends on, it must be possible to account for logical correctness in an environment that tolerates many different theories, some incompatible with others. Justifications for the validity of arguments depends on the interconnections of objects and the reliability of the tools used to create them. We do not believe there is any absolute criteria to be applied, but that instead it should be the user who determines what counts as a valid form of justification.

Large Scale Object Management: To facilitate integration of multiple theories and to avoid corruption the system should use abstract object references instead of conventional naming schemes. Users naturally expect to be able to rerun proofs of previously established theorems. A persistent store if required if established results are to be immune to changes and extensions of existing theories or to improvements in methods.

These design principles will be discussed in more detail below.

3 Results

3.1 Readable Proofs

One aspect of the project has been the linking and cross-referencing two kinds of libraries, informal text-based accounts and formal proofs. We have already explored some of the benefits of connecting formal content to text. Our web libraries [19] provide a number of examples of how we have done this, and our articles [20, 10, 4] describe the process and potential applications. Our experiments in education give examples of the use. Mixing formal and informal presentations encourages people to take steps toward precision and formal methods. It allows them to make informal inferences that connect formal concepts. It allows formal tools to support logical connections between sections of mixed material.

In the final paper we will use material from Stuart Allen's library of discrete mathematics [2] to illustrate the importance of having readable formal proofs. We will show how detail can be progressively exposed, how readings can be linked to the proof and how the proof can we interlinked to informal accounts, called a "gloss" in Allen's presentations.

3.2 Manipulating Formal Objects

We will discuss in the full paper the mechanisms by which material from the Nuprl libraries can be automatically posted to the Web in HTML. We will also illustrate some of the operations that we regularly perform on theorems, definitions and proofs.

3.3 Semantic Processing of Formal Mathematics

Our proposed formal digital library represents a significant elaboration of digital libraries of text, and the impact of work on digital libraries of formal content can be gaged in part by comparison to emerging digital libraries of text.

The basic problem with text libraries is that the material is not active in that the algorithms mentioned in it cannot be executed and the proofs cannot be checked, i.e. the contents are not *interactive*. This is because there is no effective nor obvious way to assign meaning to the text or to the abstract syntax or the XML. For example, there is no way to search the contents semantically or process any of the objects based on their semantics, for example finding a theorem that "means the same as theorem X." There is no way to process objects based on logical dependencies among them. Our methods allow such processing, and we are exploring specific information services that can be based on them.

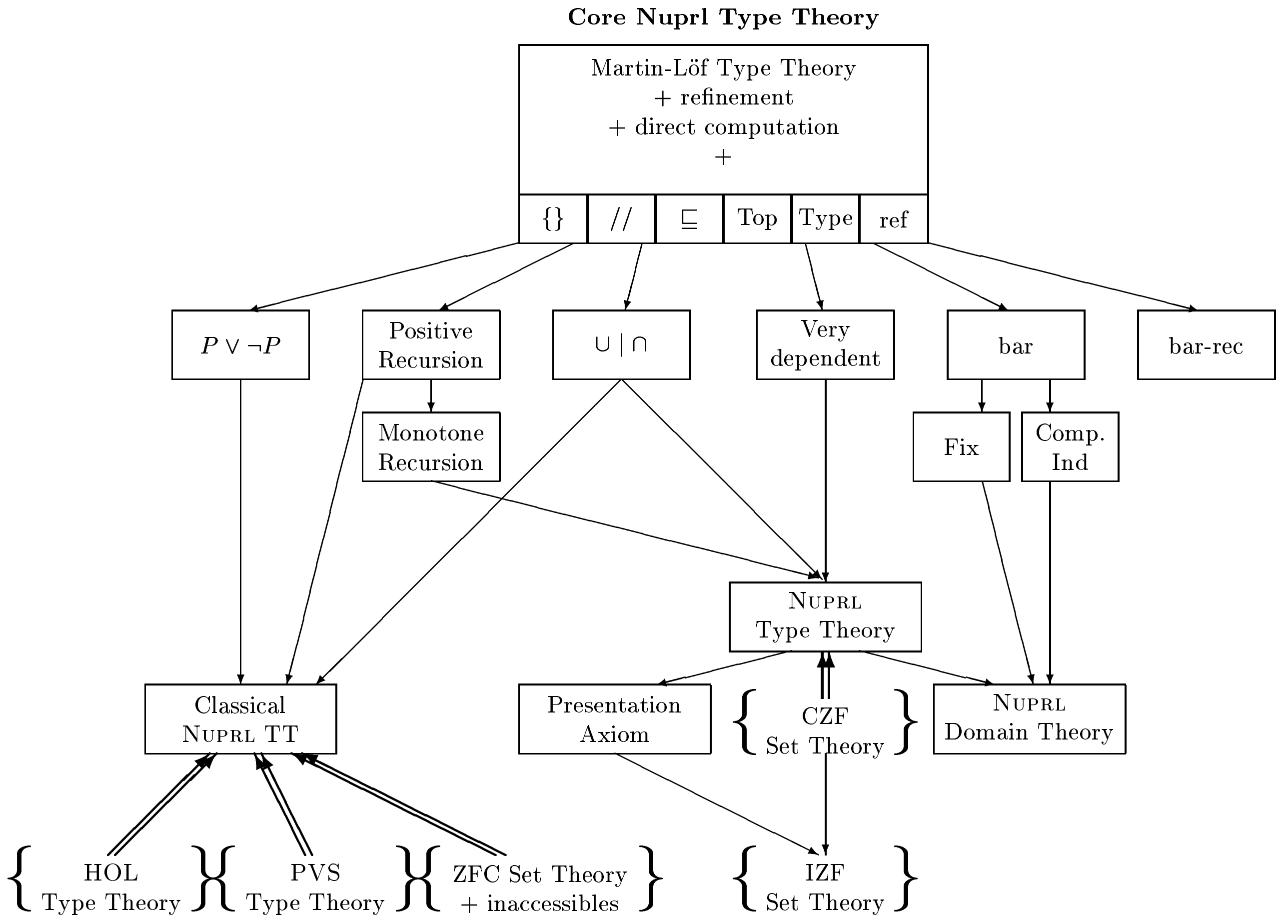

Fig. 1. Richness of Nuprl's type theory. Single arrows describe extensions of the core type theory, double arrows denote embeddings of theories.

The generality of logical methods is key to this endeavor. They support programming practice by providing *logical version control*. Indeed we currently use our Library in this way to some extent to manage Nuprl.

3.4 Properties of Formal Libraries

One way to frame the issues involved in building a logical library is to think about the problem of soundness as we change scale – from local to global. The starting point is a single theorem prover proving a single theorem in the context of a small list of previously proved theorems. Even here there are interesting logical questions, many of them were solved by the tactic mechanism which reduces a complex inferences to primitive proof steps; these can be double checked by executing a simple proof checker on the primitive proof (systems that allow very simple checkers are said to satisfy the *deBruijn principle*). Consider the scale in which multiple provers are contributing to the development of a theory (or the hardening of a software system); each is using a local context of a global library, each is adding new tactics that execute in a local environment. How do we know that the combined work is logically sound? How do we know the results will reprove as the environments are extended? How do we make the replay of proofs stable under extension?

Our conceptual path to the library design grew from our project's own need to maintain a flexible development method, permitting divergent partially independent developments, and yet to be able to justify claims of validity, exposing the assumptions of such justifications. Our old system organization encouraged the development of one basic sound series of extensions to the semantics and rules, with a few rather isolated variations which could not, without great expense, be incorporated back into the project's main line of development. Often these variations would develop significant generally useful bodies of math which could not be reliably or inexpensively exported and put into the main line. Similarly, sometimes a variant might need to delete incompatible parts of the standard, again practically putting it beyond the useful reach of those using the standard library. The need to modify standard tactic code in the variant developments further inhibited their eventual incorporation into the main line. We needed to develop a library that could contain and account for these various partially incompatible, partially independently developed contents.

Additionally, we desired to "harden" our tactic proofs so that we could rerun them at will however the library might grow, and to allow for variant refiners and independent proof checkers; and yet eventually one cannot afford to recheck every theorem whenever something has been changed. Once the library is large, most changes must be incremental, and one must be able to target the small part that must be rechecked after a modification. Ultimately, we found that our arguments for correctness, of the form, "the way this proof occurs in this library entails that it is valid", simply had too many gaps, and we proceeded to make it possible to fill them in. The larger and more complex the body of theorems becomes, the more one must rely on such arguments in order to have any confidence in the correctness of ones work. The possibility of such an argument is one of the great values of formalization in the first place [11], and we must not give it up simply because we require flexibility and naturalness of expression.

In the full paper we will present examples that forced us toward the implementation of abstract object identifiers and a persistent store. Sometimes surprising unsoundnesses are introduced into a theory by non-logical library related issues related to name spaces and aliasing.

4 Conclusions

The major technical challenges in providing library services arise from features special to formal logical libraries, mainly:

- *Theories contain code* in the form of tactics which is used in building the formal objects. We need to be able to reliably re-run this code in order to check or rebuild objects. This creates a need to be able to run the code in an environment linked to the library.
- *Dependencies* arise from logical connections among objects. Some dependencies are created by executing tactic code which makes it hard to track them.
- Theories can be very large and might take days to recheck, so there must be a way to *perform local checks and rebuilds*. Also, there is a pressing need for making the tools faster.
- The theorems in the library can be created by various tools in different logical theories with different foundations. We need to be able to express relations between logical theories in order to be able to translate theorems between different theories.
- It is essential to formalize both the object level theories and aspects of the meta-level, this leads to questions about *reflection* and *meta-level reasoning* which are some of the deepest aspects of formal systems.

References

1. Mark Aagaard and Miriam Leeser. Verifying a logic synthesis tool in Nuprl. In Gregor Bochmann and David Probst, editors, *Proceedings of Workshop on Computer-Aided Verification*, pages 72–83. Springer-Verlag, June 1992.
2. Stuart Allen. Discrete math lessons. `http://www.cs.cornell.edu/Info/People/sfa/Nuprl/eduprl/Xcounting_intro.html`, 2001.
3. Stuart Allen, Robert Constable, Richard Eaton, Christoph Kreitz, and Lori Lorigo. The Nuprl open logical environment. In D. McAllester, editor, *17^{th} International Conference on Automated Deduction*, volume 1831 of *Lecture Notes on Artificial Intelligence*, pages 170–176. Springer Verlag, 2000.
4. Mark Bickford and Jason Hickey. Predicate transformers for infinite state automata. 1999.
5. Ken Birman, Robert Constable, Mark Hayden, Jason Hickey, Christoph Kreitz, Robbert van Renesse, Ohad Rodeh, and Werner Vogels. The horus and ensemble projects: Accomplishments and limitations. In *DARPA Information Survivability Conference and Exposition (DISCEX 2000)*, pages 149–161, 2000.

6. Samuel R. Buss, Alexander S Kechris, Anand Pillay, and Richard A Shore. The prospects for mathematical logic in the twenty-frst century. *Bulletin of Symbolic Logic*, 7(2), June 2001.
7. 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, and S. F. Smith. *Implementing Mathematics with the Nuprl Development System*. Prentice-Hall, NJ, 1986.
8. 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.
9. Robert L. Constable. Types in logic, mathematics and programming. In S. R. Buss, editor, *Handbook of Proof Theory*, chapter X, pages 684–786. Elsevier Science B.V., 1998.
10. Robert L. and Paul B. Jackson Constable, Pavel Naumov, and Juan Uribe. Constructively formalizing automata. In *Proof, Language and Interaction: Essays in Honour of Robin Milner*, pages 213–238. MIT Press, Cambridge, 2000.
11. Gottlob Frege. Begriffsschrift, a formula language, modeled upon that for arithmetic for pure thought. In *From Frege to Gödel: A Source Book in Mathematical Logic, 1879-1931*, pages 1–82. Harvard University Press, Cambridge, MA, 1967.
12. Amanda Holland-Minkley, Regina Barzilay, and Robert L. Constable. Verbalization of high-level formal proofs. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence*, pages 277–284. AAAI, July 1999.
13. Douglas J. Howe. The computational behaviour of Girard's paradox. In *Proceedings of the Second Symposium on Logic in Computer Science*, pages 205–214. IEEE, June 1987.
14. Paul B. Jackson. *The Nuprl Proof Development System, Version 4.1 Reference Manual and User's Guide*. Cornell University, Ithaca, NY, February 1994.
15. Xiaoming Liu, Christoph Kreitz, Robbert van Renesse, Jason Hickey, Mark Hayden, Kenneth Birman, and Robert Constable. Building reliable, high-performance communication systems from components. In *17^{th} ACM Symposium on Operating Systems Principles (SOSP'99)*, volume 34 of *Operating Systems Review*, pages 80–92, 1999.
16. Chetan Murthy. A computational analysis of Girard's translation and LC. In *Proceedings of Seventh Symposium on Logic in Comp. Sci.*, pages 90–101, 1992.
17. Chetan Murthy. Finding the answers in classical proofs: A unifying framework. In *Informal Proceedings of the Second Workshop on Logical Frameworks II*, pages 271–290, 1992.
18. Nuprl web pages. `http://www.cs.cornell.edu/Info/Projects/NuPrl/nuprl.html`.
19. Nuprl libraries web pages. `http://www.cs.cornell.edu/Info/Projects/NuPrl/Nuprl4.2/Libraries/Welcome.html`.
20. Nuprl publications. `http://www.cs.cornell.edu/Info/Projects/NuPrl/html/publication.html`.
21. Stephan Schmitt, Lori Lorigo, Christoph Kreitz, and Alexey Nogin. JProver: Integrating connection-based theorem proving into interactive proof assistants. Technical report, 2001.